PHẠM HUY HOÀNG

CODE DẠO KÝ SỰ

Lập trình viên đâu phải chỉ biết code

PHẠM HUY HOÀNG CODE DẠO KÝ SỰ

trithuctrebooks

Trithuctrebooks
CÔNG TY TNHH SÁCH VÀ TRUYỀN THÔNG VIỆT NAM
ĐC: Số nhà 23/56/376 đường Bưởi - P.Vĩnh Phúc - Q.Ba Đình - Hà Nội
ĐT: 04. 6293 2066 - Fax: 04. 3838 9613
E-mail: trithuctrebooks@gmail.com

CODE DẠO KÝ SỰ
Giá: 159.000đ
ISBN: 978-604-943-417-4





PHẠM HUY HOÀNG

CODE DẠO KÍ SỰ
LẬP TRÌNH VIÊN ĐÂU PHẢI CHỈ BIẾT CODE

LẬP TRÌNH VIÊN ĐÂU PHẢI CHỈ BIẾT CODE

NHÀ XUẤT BẢN TRI THỨC

LỜI GIỚI THIỆU

Chào các bạn, mình là Phạm Huy Hoàng, chủ một blog IT mang tên Tôi đi code dạo.

Hiện nay, ngành IT nói chung và lập trình nói riêng đang trở thành một ngành hot, được khá nhiều bạn sinh viên lựa chọn. Tuy nhiên, so với nước ngoài, các bạn sinh viên Việt Nam chịu khá nhiều thiệt thòi vì thiếu những tấm gương và tài liệu để học hỏi. Thuở còn là sinh viên, mình cũng từng có những thắc mắc, trăn trở về kĩ thuật, về con đường nghề nghiệp, nhưng không có ai giải đáp.

Là một lập trình viên, các bạn cần học rất nhiều, nhưng không sách vở nào nói về cách tự học cho hiệu quả. Lập trình viên cần biết cách giao tiếp và làm việc nhóm, nhưng ít thầy cô nói cho các bạn biết điều này. Lập trình viên cần phải giỏi tiếng Anh, nhưng hầu như đi làm rồi các bạn mới tự nhận ra.

Không biết những điều này, bạn sẽ phải hứng chịu vô số gạch đá trên con đường nghề nghiệp. Do vậy, chúng ta cần những đầu sách định hướng nghề nghiệp và những kĩ năng phải có của người lập trình viên.

Tuy nhiên, đa phần sách cho dân IT hiện nay quá tập trung vào kĩ thuật và công nghệ (kĩ năng cứng), quên mất những kĩ năng mềm mà lập trình viên nên có. Những quyển sách trên cũng khá hàn lâm và khô cứng, khó tiếp thu.

Cuốn sách này không như thế! Vậy nó có gì hot?

LẬP TRÌNH VIÊN ĐÂU PHẢI CHỈ BIẾT CODE

Đây là cuốn sách duy nhất tập trung vào phần kĩ năng mềm mà mỗi lập trình viên cần có. Đi kèm với chúng những kĩ năng cứng được đúc kết qua kinh nghiệm bao năm làm việc của tác giả. Sách đảm bảo sẽ đưa bạn đọc từ mềm đến cứng.

Thay cho những chương sách dày cộm toàn chữ, nội dung sách được chia làm nhiều bài viết ngắn gọn, mỗi bài viết đề cập đến một khía cạnh khác nhau. Giọng văn ngắn gọn, hài hước dí dỏm, đọc không hề cứng nhắc như sách kĩ thuật mà lại rất dễ tiếp thu. Đoạn này không phải nhận xét của mình mà đó là nhận xét chung của khoảng 2000 bạn đọc ghé thăm blog mỗi ngày.

Ngành lập trình rất rộng lớn, không thể đề cập hết trong một cuốn sách. Do vậy, mình tập trung nhiều vào việc rèn luyện khả năng tự học và định hướng cho bạn đọc. Có kĩ năng tự học, có định hướng tốt, bạn sẽ dễ dàng sống sót và thăng tiến trong ngành này.

MỤC LỤC

TÓM TẮT NỘI DUNG

Nội dung cuốn sách gồm 3 phần chính:

- Phần 1 tập trung vào những kĩ năng mềm và thái độ mỗi lập trình viên cần có trên ba quãng đường: Thuở còn ngồi ghế nhà trường, khi sắp ra trường và khi bắt đầu đi làm. Tuy vậy, các bạn sinh viên có thể đọc phần "làm việc" để hiểu thêm về công việc tương lai, cũng như các bạn đã đi làm có thể đọc phần "học hành" để biết và bổ sung những kĩ năng mình còn thiếu.
- Phần 2 đi sâu hơn về những kĩ thuật lập trình từ cơ bản đến nâng cao. Đa phần những kĩ thuật này không được dạy hoặc chỉ được dạy khá sơ sài ở nhiều trường đại học, dẫn đến việc sinh viên phải tự học, tự mò mẫm khi đi làm.
- Phần 3 là những mẩu chuyện và trải nghiệm nho nhỏ của chính tác giả trong quãng thời gian làm lập trình viên ở trong và ngoài nước.

Đối tượng chính của sách là các em lớp 12 sắp chọn ngành IT, các bạn sinh viên IT, những bạn lập trình viên vừa ra trường mới đi làm, và những bạn trẻ muốn tìm hiểu về ngành IT. Do vậy, sách không tập trung quá nhiều vào kĩ thuật (ngoại trừ phần 2 nặng về kĩ năng lập trình).

Bài viết trong sách sử dụng nhiều ví dụ sinh động, ngôn từ dễ hiểu, không hàn lâm những nên bạn đọc không có chuyên môn về IT cũng có thể thoải mái đọc và thưởng thức. Những từ ngữ thông dụng trong ngành IT sẽ được liệt kê phía cuối sách, giúp bạn đọc dễ tìm hiểu hơn.

PHẦN 1 – KĨ NĂNG MỀM MỀM

Đa phần các bạn sinh viên thường nghĩ rằng chỉ cần học giỏi, vững kĩ năng cứng (kĩ năng lập trình) thì sẽ dễ dàng kiếm được việc làm, thăng tiến. Đây là một suy nghĩ khá sai lầm, vì đôi khi kĩ năng mềm nhiều khi còn quan trọng hơn kĩ năng cứng rất nhiều lần.

Nếu bạn code giỏi nhưng không biết giao tiếp với trưởng nhóm và các thành viên khác, bạn sẽ không thể truyền đạt ý kiến của mình hay lãnh đạo. Nếu bạn lập trình tốt nhưng không rành tiếng Anh, không biết tự học thì kiến thức của bạn sẽ rất nhanh hết thời, làm bạn bị tụt hậu. Hoặc giả bạn có giỏi đến mấy nhưng nếu cứ mang thái độ "mình là sinh trường A, B danh giá, giỏi hơn hẳn bọn kia!" đi xin việc, bạn sẽ rớt ngay từ vòng gửi xe, à không, gửi nón.

Do vậy, mình dành ra phần đầu cuốn sách để tập trung vào những kĩ năng mềm mà lập trình viên cần có, nên có và phải có để trở thành một lập trình viên (developer) chuyên nghiệp.

Giai đoạn 1 – Học hành

Đây là giai đoạn bạn cần tập trung học các kiến thức nền tảng trong trường, rèn luyện khả năng tự học, tiếng Anh v…v. Các bài viết trong phần này sẽ mang tính định hướng, đồng thời đề cập tới những kĩ năng nói trên.

VÀI LỜI KHUYÊN VÀ ĐỊNH HƯỚNG CHỌN TRƯỜNG CHO CÁC BẠN TRẺ

Chọn trường đại học và chọn ngành học là một ngưỡng cửa khá quan trọng của cuộc đời. Có lẽ đa phần bạn đọc của sách là sinh viên đại học, hoặc đã đi làm nên chắc sẽ không cần đọc bài này. Tuy nhiên, hãy đưa nó cho em/cháu bạn hoặc phụ huynh của các em. Bài viết sẽ giúp họ có cái nhìn đúng hơn trong việc chọn trường, giúp các em hiểu hơn về công việc mình sẽ làm trong tương lai.

Lời khuyên đầu: Học trường vừa sức

Lời khuyên đầu tiên mình muốn gửi tới các bạn và các em là: Chọn trường vừa sức mình. Vừa sức ở đây không chỉ có nghĩa là vừa sức đậu, mà còn có nghĩa là vừa sức học và cạnh tranh.

Tại sao lại chọn trường vừa sức mà không phải là trường nổi tiếng? Theo lẽ thường, chất lượng dạy và học ở của các trường nổi tiếng này khá cao, tấm bằng đại học danh tiếng cũng rất có ích khi bạn vừa ra trường xin việc hoặc muốn tiếp tục học lên cao.

Tuy nhiên, ở các trường này, do chất lượng đầu vào cao nên bạn sẽ phải học hành và cạnh tranh với những bạn bè giỏi hơn (còn được gọi dưới cái tên thiên tài hay quái vật). Nếu không đủ giỏi, việc cạnh tranh với những thành phần này dễ làm bạn nản lòng thoái chí. Chưa kể, do các giáo viên đã quen với việc dạy dỗ học sinh thông minh, có thể họ sẽ giảng giải với tốc độ nhanh hơn, khó hiểu hơn, làm bạn khó theo kịp.

Ngoài ra, vào những trường giỏi, bạn rất khó để vào top đầu lớp hoặc gây ấn tượng với giáo viên (vì học sinh giỏi nhiều quá rồi). Ngày xưa, mình cũng đậu đại học BK HCM nhưng không học, một phần là do FPT có học bổng 70%, một phần là do mình không muốn bỏ quá nhiều công sức vào việc cạnh tranh học tập. Vào FPT, mình dễ dàng đứng đầu lớp, được nhiều giáo viên thương và để ý. Nhờ vậy, mình dễ dàng xin thư giới thiệu của họ khi làm đơn du học.

Lời khuyên thứ hai: Tự học đi, không ai dạy bạn đâu!

Lời khuyên thứ hai là: Kiến thức ở trong trường không đủ để bạn xin việc đi làm đâu[1]. Do đó hãy bắt đầu rèn luyện kĩ năng tự học và tìm hiểu từ bây giờ đi. Những kiến thức bạn có được khi học đại học chỉ là nền tảng thôi, khó mà áp dụng ngay vào công việc! Nếu chỉ biết những gì được dạy mà không biết tự mày mò học thêm, bạn sẽ gặp khá nhiều khó khăn khi ôm mớ nền tảng đó đi xin việc đấy!

Nhiều bạn sinh viên đi học được một, hai năm thì bắt đầu rơi vào tình trạng mất phương hướng vì cảm thấy những thứ mình học quá vô dụng, chẳng làm được gì. Thay vì chờ được thầy cô dạy, hãy tận dụng những kiến thức nền tảng đã có để tự học, sau đó áp dụng những thứ vừa học để tạo ra một sản phẩm gì đó, bạn sẽ thấy thích học ngay thôi. Còn nữa, nhớ phải tập trung trau dồi tiếng Anh nhé[2]. Học IT mà không giỏi tiếng Anh thì khó phát triển lắm đấy!

Học IT xong thì ra làm gì?

Ở Việt Nam, hầu như mọi người chỉ biết ngành IT (Information Technology – Công nghệ thông tin), chứ không biết tường tận ngành đó làm những gì. Do đó, mình sẽ giải thích một số chuyên ngành của ngành IT, về những thứ bạn sẽ học cũng như công việc bạn sẽ làm sau khi ra trường.

Hiện tại ngành IT có một số chuyên ngành sau:

- **Khoa học máy tính (Computer Science)**: Bạn sẽ học các thứ liên quan tới cách thức máy tính hoạt động. Theo như tên gọi "Khoa học", chuyên ngành này thường nặng về nghiên cứu.
- **Kĩ nghệ phần mềm (Software Engineering)**: Ngành này cũng học một số môn tương tự như CS. Tuy nhiên, chuyên ngành này thiên về thực tế và xây dựng phần mềm nên bạn được học thêm 1 số ngành như: Qui trình phát triển phần mềm, Kiểm thử phần mềm. Học ngành này bạn có thể viết ứng dụng,

[1] Đọc kĩ hơn trong bài viết: Những điều trường đại học không dạy bạn"

[2] Mình có chia sẻ kinh nghiệm học và thi trong bài "Tôi đã học tiếng Anh như thế nào"

viết website hoặc xây dựng một hệ thống. Sinh viên tốt nghiệp cả hai ngành CS và SE đều có thể làm lập trình viên (developer).

- **Hệ thống thông tin (Information System)**: Ngành này thiên về phân tích thiết kế hệ thống dựa theo yêu cầu của khách hàng. Bạn sẽ phải học một số môn liên quan tới Thương mại điện tử, cách thức các doanh nghiệp hoạt động. Khi ra trường bạn có thể làm ở vị trí Business Analyst (BA).
- **Hệ thống nhúng (Embedded System)**: Ngành này tập trung vào việc xử lý tín hiệu số, thiết kế mạch điện, chip điện tử và linh kiện. Khi ra trường, bạn cũng là lập trình viên, nhưng lập trình cho các thiết bị hoặc mạch điện. Ngành này hơi khó và khô khan hơn ngành SE, nhưng lương trung bình cao hơn một chút.
- **Lập trình mạng (Network Engineering)**: Ngành này dạy về cơ sở hạ tầng mạng, cách lắp đặt hệ thống, v…v. Mấy bác tốt nghiệp ngành này là những người cài win dạo, bấm cáp dạo, sửa modem, quản lý server, thường gọi là IT Helpdesk. Họ là những người hùng thầm lặng, giúp hệ thống hoạt động trơn tru.
- **An toàn thông tin (Infomation Security)**: Ngành này tập trung về bảo mật, bạn sẽ được học về kiến trúc hệ thống, mã hóa, bảo mật, những phương thức hack và cách phòng chống. Ngành này phù hợp với những bạn hâm mộ các anh hacker. Ra trường, bạn có thể làm hacker mũ trắng hoặc chuyên viên bảo mật cho các công ty.

Có một số môn như Hệ điều hành, Mạng máy tính, Mã máy, Thuật toán, Cấu trúc dữ liệu…. mà sinh viên chuyên ngành nào cũng phải học. Giữa các trường đại học, chương trình học của các chuyên ngành này sẽ có đôi chút khác biệt.

Tóm tắt:

- Nên chọn trường vừa sức để bạn có thể nằm trong tốp đầu lớp
- Cần rèn luyện khả năng tự học. Trong ngành này, tự học là chính, kiến thức trong nhà trường là không đủ
- Tuỳ vào ngành học mà sinh viên IT ra trường có thể làm rất nhiều nghề: lập trình viên, quản trị mạng, an toàn thông tin,…

HAI SAI LẦM LỚN NHẤT TRONG QUÁ TRÌNH HỌC LẬP TRÌNH

Bài viết này nói về hai sai lầm lớn nhất trong quá trình học lập trình. Qua trao đổi với các bạn sinh viên, mình nhận thấy có khá nhiều bạn sinh viên mắc phải những sai lầm này (cá nhân mình hồi năm nhất năm hai cũng thế). Do đó, bài viết này sẽ cảnh tỉnh một số bạn, đồng thời chia sẻ chút kinh nghiệm để tránh các bạn đi theo vết xe đổ của mình ngày trước.

Câu nói hay gặp – trường em không dạy A, B, C ...

Dưới đây là một mẫu đối thoại giữa mình và một bạn sinh viên (giấu tên)

- Bạn: Anh ơi, học C với C++ ra trường thì làm được gì anh?
- Mình: Làm hệ thống nhúng hoặc game em nhé, lương khủng lắm đấy.
- Bạn: Em thích làm Web hoặc làm app di động cơ, C++ làm được không anh?
- Mình: Không em nhé, muốn làm web thì học HTML/CSS/JS. Sau đó có thể chuyển qua làm hybrid app mobile [3], hoặc học Android để viết app.
- Bạn: Mấy cái đó trường em không dạy anh ơi!!!
- Mình: ...

Một câu mình nói mình hay được nghe các bạn nói là: trường em chỉ dạy C, trường em chỉ dạy Java, mấy thầy cô không dạy HTML hay làm Web… Mình đã từng nói ở đầu sách, đại học chỉ cho bạn các kiến thức nền tảng về lập trình. Họ sẽ không dạy bạn cách code như thế nào, cách làm việc, cách sử dụng một ngôn ngữ hoặc framework ra sao, mà bạn sẽ phải tự dạy mình!!.

Thái độ trường không dạy nên không biết là một thái độ học tập cực kỳ sai lầm. Vấn đề không phải người ta dạy cho bạn cái gì, mà là bạn có thể học được cái gì! Thái độ ngồi chờ sung rụng, có người dạy mới học này sẽ cản trở bạn trên con đường tìm hiểu cái mới, tự cập nhật

[3] Xem thêm trong bài "Tổng quan về lập trình ứng dụng di động"

kiến thức cho bản thân. Nếu giữ thái độ này, kiến thức của bạn sẽ nhanh chóng lạc hậu, lỗi thời, gây ra nhiều khó khăn trên con đường thăng tiến của bạn.

Học tập thế nào cho đúng??

Việc học phải mang tính chất chủ động chứ không phải là bị động. Bạn phải tự tìm cái cần học và tự sắp xếp thời gian để học. Bạn nào kiến thức vững, đủ kiên nhẫn để tự học thì có thể tải ebook tiếng Anh hoặc tìm nguồn tự học trên mạng. Bạn nào kiến thức còn yếu thì có thể ra trung tâm để có người kèm cặp.

Nhà tuyển dụng chỉ cần biết bạn có kiến thức hay không và có làm được việc hay không. Họ không quan tâm là kiến thức đó bạn có được từ nhà trường, từ trung tâm hay do tự học. Thứ duy nhất bạn phải nhớ là: thầy cô hay trung tâm cũng không thể vá lỗ hổng kiến thức hay dạy cho bạn tường tận được, mà chính bạn mới là người nỗ lực hấp thu, biến kiến thức của họ thành kiến thức của mình.

Chưa kể, chương trình học ở các trường bây giờ… cũ xì, quanh đi quẩn lại chỉ có WinForm, WebForm, Java Servlet… . Nếu cứ "há miệng chờ sung, dạy gì học nấy", bạn sẽ không có đủ kĩ năng cần thiết để xin việc khi ra trường đâu nhé!

Ngoài ra, đừng nghĩ rằng chỉ học một lần cho biết là xong, nguy hiểm lắm! Công nghệ liên tục thay đổi, bạn cũng phải thường xuyên cập nhật kiến thức bản thân. Trước đây mình từng có khoảng 2 năm kinh nghiệm lập trình C#. Đầu năm nay, lúc mình xem lại thì công nghệ đã được cập nhật, làm cho kiến thức cũ của mình lỗi thời hết cả! Thay vì than trời trách đất, mình đành phải tự học để cập nhật kiến thức mới thôi.

Sai lầm thứ hai: Cẩn thận, chưa chắc học nhiều/xem nhiều là sẽ giỏi!!

Người Việt chúng ta có thói quen ghét ai ghét cả đường đi lối về. Khi đã tin tưởng hay thần tượng ai đó thì nó nói gì cũng tin; khi đã ghét thằng nào thì nó nói gì cũng sai, cũng nhảm nhí. Thái độ này dễ làm bạn tiếp nhận sai tiếp nhận thông tin sai cách!

Mình hay đọc sách Tony Buổi Sáng, có những bài viết về cách nhìn cuộc sống khá hay. Thay vì hâm mộ, nuốt từng câu từng chữ của "dượng", vẫn có những đoạn văn chém gió, những cách nhìn mà mình không đồng tình. Tuy vậy, mình vẫn chắt lọc những điều hay, những điều tác giả muốn gửi gắm, còn mấy vấn đề mình không đồng tình thì bỏ qua.

Tương tự, các bạn nên đọc sách, nghe thầy cô giảng nhưng đừng nên tin tưởng hoàn toàn những gì được nghe. Hãy tự hỏi xem: Thầy cô hay sách nói như vậy đúng hay sai, có chứng cứ gì không? Cuộc sống có câu "Không có gì là vĩnh cửu", có thể bây giờ mình cho điều đó là đúng, nhưng trong tương lại điều đó lại sai thì sao?

Đừng tin toàn bộ những gì sách nói, cũng đừng nuốt từng câu từng lời của thầy cô hay tác giả. Nghe người khác nói cái gì cũng phải nghi ngờ vả kiểm chứng. Hãy xem tác giả là một thanh niên đang chém gió với mình thông qua sách, cái gì đúng thì gật gù đồng ý, cái gì sai thì phản bác lại ngay.

Ngoài việc học nhiều, bạn còn phải biết cách lọc bỏ, chọn lựa những thông tin có ích cho bản. Những gì hay thì hãy ghi nhớ và học theo; những gì nhảm nhí thì cứ bỏ qua, coi như nó không tồn tại. Nói một cách dân dã là phải biết cách "gạn đục khơi trong" từ vô số nguồn kiến thức.

Kết luận

Sửa được hai sai lầm về thái độ học tập bị động và chọn lọc kiến thức, bạn sẽ thấy mình trở nên vô cùng tự tin. Công nghệ A/B không có trong chương trình học? Chả sao, chỉ cần tự học vài buổi là xong! Càng học nhiều, bạn sẽ càng thấy việc học cái mới trở nên rất dễ dàng và nhanh chóng.

Hi vọng, sau bài viết này, mình sẽ không còn phải nghe câu "em không biết cái ABC này, trường với thầy cô không dạy" nữa. Thay vào đó, mình hi vọng sẽ được nghe câu: "Em đang tự tìm hiểu cái ABC, anh chỉ em một số nguồn học và những điều cần lưu ý nha".

LẬP TRÌNH VIÊN ĐÂU PHẢI CHỈ BIẾT CODE

Tóm tắt:

- Đừng trong chờ vào việc nhà trường sẽ dạy cho bạn kiến thức để làm việc. Chịu khó tự học càng sớm càng tốt.
- Đọc nhiều, học nhiều là tốt, nhưng phải biết cách loại bỏ những thứ vô bổ, giữ lại những điều có ích.

Giai đoạn 2 – Ra đời

Đây là giai đoạn bạn gần ra trường, sắp đi làm. Ở giai đoạn này, do đã có kiến thức nền tảng vững từ những môn học ở trường, bạn cần tập trung tìm hiểu thêm về ngành nghề, đồng thời tự trang bị những kĩ năng cần có để xin việc.

HỌC NGÔN NGỮ LẬP TRÌNH NÀO BÂY GIỜ?

Đây một câu hỏi mà mình thường nhận được từ các em sinh viên mới ra trường, mới vào đại học, hoặc chưa biết gì về lập trình: "Giờ mình nên học ngôn ngữ lập trình nào đây?".

Nghe đơn giản, nhưng đây là một câu hỏi có độ khó khá cao, sánh ngang với câu "Em nên làm nghề gì, học đại học nào?" của các em học sinh cấp 3. Trong phạm vi bài viết này, mình sẽ đưa ra một số dữ liệu tham khảo và lời khuyên cá nhân.

Trước khi hỏi câu này, hãy tự hỏi : Mình muốn học lập trình để làm gì?

Khi được hỏi "Giờ mình nên học ngôn ngữ lập trình nào đây?", mình luôn hỏi lại câu này "Bạn/Em muốn học lập trình để làm gì?". Trả lời được câu hỏi này, bạn đã xác định được 50% ngôn ngữ mình cần học. Dưới đây là một số câu trả lời mình hay nhận được.

1. Em vừa ra trường, trường chỉ dạy C, C++, … giờ em cần học ngôn ngữ gì để dễ kiếm việc làm, lương cao?

Thị trường việc làm IT hiện tại khá rộng, tạm chia làm 3 mảng: embedded (lập trình nhúng), web và mobile. Thị phần mảng Game khá nhỏ nên mình không nhắc đến.

- **Mảng embedded**: yêu cầu khá cao về trình độ, sử dụng ngôn ngữ lập trình C, C++, có thể dùng Java. Nếu bạn là lập trình viên C++ cứng, mức lương rất khá và mức độ cạnh tranh cũng ko nhiều.
- **Mảng mobile**: Chiếm thị phần cao nhất vẫn là app cho Android viết bằng Java, tiếp theo là app cho IOS, viết bằng Objective-C[4]. Java là một ngôn ngữ khá dễ học, độ phổ biến cũng cao, ứng dụng rộng. Với kiến thức Java bạn cũng có thể chuyển qua mảng Web.
- **Mảng web**: Để có thể trở thành lập trình viên Web, bạn phải biết lập trình front-end (Dùng HTML/CSS và ngôn ngữ

[4] Xem kĩ hơn trong "Tổng quan về lập trình ứng dụng di động"

JavaScript). Sau đó học một ngôn ngữ lập trình back end (C#, Java, PHP)[5].

Nếu muốn học để kiếm tiền, hãy xác định mình muốn làm mảng công việc nào, sau đó tìm các khảo sát trên mạng và nghiên cứu mức lương trung bình của developer ngôn ngữ đó.

Hiện tại, có khá nhiều PHP developer nên lương trung bình của PHP developer hơi thấp so với số còn lại. Ngoài ra, một số ngôn ngữ như Rails, Python,… ít người học, developer hiếm nên biết các ngôn ngữ này sẽ có thu nhập khá hơn.

2. Mình muốn làm website hoặc ứng dụng cho người nhà, bản thân v….v.

Đây là câu trả lời mình nhận được từ một số bạn học tài chính ngân hàng, kinh tế,… Nếu bạn muốn làm một ứng dụng di động, Java là lựa chọn tốt nhất. Để tạo website, hiện tại có rất nhiều hướng dẫn tạo website bằng Joomla, Drupal, Wix,… mà ko cần kiến thức lập trình. Các bạn có thể học thêm PHP để có thể tùy biến, thêm tính năng cho trang web.

Lựa chọn thật ra không quan trọng, học một ngôn ngữ mới là chuyện đơn giản

Đọc tới đây, hẳn nhiều bạn sẽ bảo rằng mình nổ, hoặc nghĩ rằng "dám chắc thằng này không phải coder, phán như thánh". Trước khi ném đá, hãy bình tĩnh nghe mình giải thích và trình bày.

Mình cũng từng là sinh viên IT như các bạn. Môn đầu tiên về lập trình mình được học khi vào đại học là: "Cơ bản lập trình với C". Mình từng điên đầu với khai báo biến, tách hàm, điều kiện, vòng lặp, IO…. Môn tiếp theo là "Lập trình hướng đối tượng với C++". Phải thú thật C++ không phải là ngôn ngữ phù hợp để học hướng đối tượng. Mình từng nhầm lẫn trước các khái niệm "tính bao đóng, tính kế thừa".

Do đó, bản thân mình cũng biết sự khó khăn gặp phải khi học một ngôn ngữ. Tuy vậy, mình vẫn khẳng định học một ngôn ngữ mới là chuyện đơn giản.

---

[5] Xem kĩ hơn trong "Kĩ năng cần có của Web Developer"

LẬP TRÌNH VIÊN ĐÂU PHẢI CHỈ BIẾT CODE

Vì sao? Hãy tự xem lại kiến thức lập trình bạn có được khi vừa ra trường:

- Học qua 1,2 ngôn ngữ gì đó
- Cấu trúc dữ liệu và thuật toán
- Thiết kế, truy vấn cơ sở dữ liệu
- Design pattern (Có thể)
- Khả năng design

Khi mới tiếp cận lập trình, chúng ta cảm thấy khó khăn vì phải làm quen với vô số khái niệm mới. Tuy nhiên, khi đã có kiến thức cơ bản, việc tiếp cận ngôn ngữ mới trở nên rất dễ dàng. Chúng ta học gì khi học một ngôn ngữ lập trình mới? Đây là câu trả lời:

- Cách khai báo hàm, biến
- Cách khai báo vòng lặp, điều kiện if/else
- Các kiểu cấu trúc dữ liệu: list, set, tuple, …
- IO, multi-thread, delegate, event
- IDE phù hợp, cách build, debug
- Các framework, cách sử dụng, ….

Nếu bạn đã biết cách viết for, if/else, while … trong Java, khi chuyển qua học C# hoặc javascript, cấu trúc hàm for, if/else… vẫn giữ nguyên. Kiến thức của bạn được kế thừa từ ngôn ngữ lập trình trước, do đó việc học sẽ diễn ra nhanh hơn. Hoặc khi bạn đã rõ cơ chế làm việc của ASP.NET RestAPI, việc học cách xây dựng RestAPI bằng Spring của Java cũng không quá khác biệt.

Mình từng tự học Python mất 1 tuần, và học framework Django mất khoảng 2 tuần nữa. Lý do mình học nhanh vậy là vì:

- Mình đã có kiến thức cơ bản về lập trình (class, data structure ….)
- Mình biết những gì mình cần học. Khi mới lập trình, bạn không biết mình cần học gì. Tuy nhiên nếu đã có kiến thức nói chung về lập trình, bạn sẽ biết mình tập trung học những gì, điều này tiết kiệm rất nhiều thời gian.
- Mình biết là mình làm được. Khi mình hỏi bạn bè chung ngành "Học một ngôn ngữ mới mất bao lâu", hầu hết đều trả lời "một

tháng hoặc hơn". Vì thấy tốn nhiều thời gian và khó khăn như vậy nên hầu như họ rất "ngại" học ngôn ngữ mới.

Đừng sợ mình sẽ chọn nhầm ngôn ngữ, cứ học đi. Việc học một ngôn ngữ lập trình mới khi bạn đó có kiến thức cơ sở khá đơn giản, không hề khó khăn và mất thời gian như bạn nghĩ. Thêm vào đó, việc biết nhiều ngôn ngữ sẽ giúp bạn có lợi thế hơn khi xin việc.

Lời khuyên của bản thân Hoàng

Dưới đây là một số lời khuyên của mình, dựa theo kinh nghiệm cá nhân (Mình chỉ có kinh nghiệm mảng web và mobile, nên các lời khuyên có thể sẽ không áp dụng được cho mảng embedded system):

...

Tóm tắt

- Trước khi hỏi "Cần học ngôn ngữ gì?", hãy tự trả lời "Học lập trình để làm gì?"
- Đừng lo chọn sai ngôn ngữ để học, học một ngôn ngữ mới rất đơn giản
- Đừng chạy theo công nghệ, hãy tập trung vào kiến thức cơ bản và những thứ lâu bền
- Nếu quá dư thời gian, hãy tập trung vào học JavaScript

MUÔN NẺO ĐƯỜNG TÌM VIỆC

Không như các ngành nghề khác, ngành lập trình là một ngành khá dễ xin việc ở Việt Nam. Chỉ cần có khả năng code kha khá, các bạn sinh viên có thể dễ dàng xin việc với mức lương tạm ổn, không cần phải quen biết, lót tay hay con ông cháu cha gì cả.

Bài viết này chia sẻ kinh nghiệm mình có được trong quá trình xin việc, từ lúc viết CV cho đến lúc phỏng vấn. Hi vọng chúng sẽ có ích cho bạn trên bước đường tìm việc.

Phần 1: Tìm việc và viết CV

Tìm việc ở đâu?

Các trang web dưới đây là những trang web về việc làm lớn nhất Việt Nam hiện tại. Đây là những kênh trung gian giữa các nhà tuyển dụng và người tìm việc, do đó mình khuyên các bạn đang tìm việc hãy tạo một CV online cho mỗi trang này:

- IT Việc: https://itviec.com/
- Linkedin: https://www.linkedin.com/
- Vietnamworks: http://topit.vietnamworks.com/
- Jobtown: https://jobtown.co/
- Một số group lập trình trên facebook cũng thường có các mẩu tin tuyển dụng.

Lâu lâu vẫn có nhận được điện thoại mời phỏng vấn của nhân sự các công ty. Mình không cần nộp CV hay ứng tuyển vì họ đã xem CV trên mạng rồi. Cá nhân mình khuyến khích các bạn nên trau chuốt CV trên LinkedIn, chức năng connect của trang này cho phép bạn kết nối với nhiều người (đồng nghiệp, cấp trên), mở rộng mạng lưới nghề nghiệp của bạn.

Viết và gửi CV

1. Nội dung

Do đã có vô số các bài viết hướng dẫn viết CV trên mạng rồi, mình sẽ không nhắc lại. Nếu bạn chưa biết trình bày CV như thế nào, cần những

thông tin gì, hãy tìm một CV mẫu cho vị trí của mình (vd bạn là junior developer hãy xem CV mẫu của 1 junor dev, tương tự với junior QC), chỉnh sửa lại cho phù hợp. Format CV của linkedin cũng khá ổn; chỉ cần bê các phần từ trên linkedin về, chỉnh sửa một chút là bạn đã có một CV khá chuẩn.

Một điều cần lưu ý: Hãy chỉnh sửa CV cho phù hợp với vị trí bạn ứng tuyển. Giả sử bạn biết cả C#, Java, Python, nhưng công việc bạn ứng tuyển đòi hỏi C# MVC, hãy để các kĩ năng và dự án liên quan tới C# lên đầu trong CV.

2. Cách trình bày

CV không cần phải đẹp lộng lẫy nhưng phải rõ ràng và ngắn gọn. Cỡ chữ nên phù hợp là 12-14, sử dụng font cơ bản như Times New Roman hoặc Arial. Các phần đề mục nên viết rõ ràng, in đậm, nhìn lướt qua có thể đọc được. Thường thường bên tuyển dụng cũng không quá bắt bẻ về mặt hình thức, nhưng các bạn nên chịu khó canh thẳng hàng thẳng lối, đồng thời soát kĩ các lỗi chính tả. Những điều nhỏ nhặt này sẽ thể hiện tính chuyên nghiệp và sự cẩn thận của bạn.

Các bạn cũng nên xuất file CV ra dưới dạng PDF để máy nào cũng mở được. CV nên đặt tên theo format "**[Tên vị trí] Tên bạn**", giúp nhà tuyển dụng dễ đọc và phân loại (Hoặc bạn đặt tên theo format mà nhà tuyển dụng đưa ra trong quảng cáo tuyển dụng).

3. Một số sai lầm hay gặp

- **Ảo tưởng sức mạnh**: Không biết vô tình hay cố ý mà trong phần kĩ năng, nhiều bạn tự đánh giá trình độ của mình là Intermediate, Expert, hoặc Master, dù chỉ mới ra trường, chỉ mới làm 1,2 cái đồ án chứ chưa làm dự án lớn nào. Nếu đã từng làm qua một công nghệ nào, các bạn nên ghi thời gian đã tiếp xúc và mức độ nắm vững công nghệ đó là đủ. Trình độ bạn ở mức nào thì người phỏng vấn sẽ tự xác định, bạn có nói mình là Master họ cũng không tin đâu.
- **Gây chú ý không đúng cách**: Dùng font màu mè lạ mắt để đập vào mắt nhà tuyển dụng. Cách này thường gây tác dụng ngược. Họ có thể quăng CV của bạn vào sọt rác không thương tiếc.

- **Chém gió**: Mình từng gặp trường hợp có bạn chém gió về số năm đi làm và công nghệ mình biết trong CV. Tới lúc phỏng vấn, bị vặn hỏi thì bạn bảo là: mình làm part time, công nghệ ABC gì đó mình hiểu nhưng chưa làm bao giờ … Nhiêu đó là quá đủ để rớt đài rồi nhỉ? Hãy nhớ rằng mục đích của CV chỉ là giúp bạn qua được vòng gửi xe, được mời phỏng vấn thôi, chứ không giúp bạn có được việc làm ngay đâu nhé.

Phần 2: Tham gia phỏng vấn

...

Tóm tắt

- Ngành IT rất dễ tìm việc, một số kênh thường dùng: itviec, linkedin, vietnamwork.
- CV là thứ đầu tiên gây ấn tượng với nhà tuyển dụng. Hãy trình bày rõ ràng, nội dung chân thật, không thổi phổng khả năng của bản thân.
- Trước khi phỏng vấn, hãy chịu khó tìm hiểu công ty, tìm hiểu thị trường và ôn lại kiến thức.
- Luôn tỏ thái độ thân thiện hòa nhã khi phỏng vấn. Sau khi kết thúc phỏng vấn, nhớ gửi email cảm ơn.

Phần 3 – Làm việc

Sau khi tốt nghiệp ra trường và vượt qua vòng phỏng vấn gian khổ, bạn sẽ được nhận vào một công ty phần mềm. Bắt đầu trong một môi trường mới, bạn sẽ khá bỡ ngỡ khi lần đầu tiếp xúc với dự án thực tế, tuân theo qui trình, làm việc theo nhóm. Hẳn bạn cũng sẽ có vài câu hỏi về nghề nghiệp, về tương lai. Những câu hỏi đó sẽ được trả lời trong phần này.

CON ĐƯỜNG PHÁT TRIỂN SỰ NGHIỆP (CAREER PATH) CHO DEVELOPER

Các bạn sinh viên còn đang học hoặc mới ra trường sẽ khó hình dung được về những vị trí và chức danh trong ngành lập trình. Bài viết này sẽ giải đáp một số thắc mắc của các bạn như:

- Mới đi làm em có chức danh gì, công việc thế nào?
- Code lâu thì lên được chức gì, lương có cao không?
- Em chỉ thích code thôi, không thích làm trưởng nhóm, em nên định hướng thế nào.

Hiểu rõ con đường nghề nghiệp của ngành developer, các bạn sẽ dễ định hình phát triển tương lai của bản thân, cũng như dồn sức vào con đường mình đã chọn.

Mình chỉ liệt kê con đường nghề nghiệp của một developer vì bản thân mình cũng là developer. Con đường của 1 tester (QA engineer) cũng có một số chức danh tương tự nhưng lên cao sẽ khác. Các chức vụ sẽ được miêu tả theo thứ tự từ thấp lên cao nhé.

Lưu ý: Mức lương đề cập trong bài dựa theo mức lương và quảng cáo tuyển dụng của các công ty ở TP. HCM, ở các tỉnh thành khác sẽ có một số chênh lệch.

Fresher/Junior Developer

Các bạn sinh viên đi thực tập hoặc mới ra trường thường được chức danh này. Kinh nghiệm của Junior Developer thường vào khoảng 6 tháng – 1 năm. Mức lương thì tùy vào khả năng của bạn, thường dao động trong khoảng 300-500$.

Do chưa có kinh nghiệm nên fresher/junior thường được các công ty đào tạo lại, vì vậy khi phỏng vấn vị trí fresher các công ty thường chỉ chú trọng khả năng suy nghĩ logic, kiến thức lập trình cơ bản và tiềm năng phát triển của bạn. Cá nhân mình thấy chương trình đào tạo Fresher của FSOFT cũng khá tốt, có dạy nhiều thứ mà bạn sẽ tiếp xúc khi làm việc.

Do chưa có kinh nghiệm nên mọi người thường không đòi hỏi ở bạn quá cao. Công việc của một junior thường là tìm hiểu dự án hiện tại,

code các module đơn giản, fix bug với sự trợ giúp và review của senior. Ở giai đoạn junior, các bạn hãy cố gắng tranh thủ học cách code, cách thức làm việc và kinh nghiệm của các bác senior đi trước.

Developer

Code được một thời gian khoảng 1-2 năm, các bạn sẽ được gọi là Developer (Nhiều bác lên thẳng vị trí Team Leader hoặc Senior tùy vào công ty). Ở giai đoạn này, bạn đã làm qua một số project, khá rành về một số công nghệ. Mức lương của developer vào khoảng 600-900$.

Phỏng vấn cho developer dĩ nhiên là khó hơn junior. Người phỏng vấn sẽ hỏi bạn về những dự án bạn đã làm, các khó khăn bạn đã gặp phải và cách giải quyết? Ngoài ra, buổi phỏng vấn sẽ tập trung vào những công nghệ bạn đã ghi trong CV. Vì developer đã có kinh nghiệm, khi đi làm các bạn sẽ không còn "được" các anh senior kèm cặp, và cũng khó mà lấy danh nghĩa junior để hỏi, nhờ vả hay mắc lỗi nữa.

Ở giai đoạn này, bạn đã được code một số module phức tạp hơn, tham gia họp, code review, thảo luận với khách hàng v…v. Đây là giai đoạn để bạn dồn nén kiến thức, kinh nghiệm và gây dựng danh tiếng để lên nấc tiếp theo trong bậc thang nghề nghiệp.

Lý thuyết là vậy nhưng thực tế, thuở làm ở FSOFT mình ở vị trí junior được khoảng một tháng rồi nhảy vào làm các công việc của developer, nhận cả những việc khó chứ không đùn đẩy gì, nhờ vậy cũng mình học hỏi được khá nhiều.

Quản lý hay kĩ thuật?

Ở giai đoạn sau, bạn đã có thể xác định con đường cho mình. Nếu muốn tập trung vào code và kĩ thuật, bạn có thể đi theo hướng kĩ thuật: Senior Developer => Technical Lead => Software Architecture. Nếu muốn làm việc với quy trình và con người, bạn nên đi theo hướng quản lý: Team Lead => Project Manager => Manager.

Ở giai đoạn đầu, lằn ranh giữa 2 con đường này khá mờ nhạt, nhưng càng lên cao lại càng trở nên rõ ràng. Các bạn có thể xem bảng tóm tắt sau:

Hướng quản lý (Management)	**Hướng kĩ thuật (Technical)**
Team Leader Bạn trở thành trưởng một nhóm nho nhỏ khoảng 3-6 thành viên. Ngoại trừ code, bạn còn phải họp hành với cấp trên, báo cáo với khách hàng, quản lý cấp dưới. Ở giai đoạn này, bạn sẽ dần học thêm một số kĩ năng lãnh đạo, kĩ năng quản lý v…v. Ở một số cong ty nhỏ, developer lâu năm và có kinh nghiệm sẽ lên team leader. Bạn vẫn còn khá nhiều thời gian code, code giỏi có thể sẽ làm thành viên trong team tôn trọng hơn. Mức lương cho team leader thường vào khoảng 1000-1500$	**Senior Developer** Sau một thời gian làm việc, bạn nắm vững, hiểu sâu và rộng nhiều công nghệ và qui trình. Ở vị trí này, ngoại trừ khả năng code "thần thánh", bạn còn phải biết đưa ra design và solution. Ngoài ra, bạn còn phải hướng dẫn chỉ bảo các em junior mới vào, cũng như tham gia code review v…v. Đôi khi senior dev cũng kiêm luôn vị trí team leader, do đó bạn cũng cần một chút kĩ năng diễn đạt và lãnh đạo. Mức lương cho Senior Developer cũng vào khoảng 1000-1500$ (hoặc hơn).
Project Manager Lên đến vị trí này, bạn sẽ có rất ít hoặc hầu như không có thời gian code. Đa phần thời gian của bạn dùng để đọc báo, lướt voz, lướt webtretho …. Đùa đấy, công việc chính của bạn bây giờ là quản lý cấp dưới, báo cáo với khách hàng và cấp trên về tiến độ dự án, lâu lâu bạn còn phải đi phỏng vấn một số ứng viên	**Technical Leader** Bạn cần hiểu biết về công nghệ sâu và rộng vì chính bạn là người lựa chọn công nghệ, qui trình… cho một dự án. Những quyết định lớn về thiết kế, cấu trúc code … sẽ do bạn chịu trách nhiệm. Ở giai đoạn này, ngoài việc technical "cứng", bạn còn phải giỏi thuyết trình, hướng dẫn, giải thích… Vì sao

để tuyển thành viên cho dự án nữa. Bạn là người quyết định sự thành bại của một dự án, do đó nếu dự án thành công bạn sẽ được thưởng một khoản bonus kha khá tùy theo chính sách công ty. Mức lương cho PM vào khoảng 1000-2000$.	á? Khi đưa ra vấn đề thì phải giải thích hợp lý, thành viên khác nó mới hiểu, nể và làm theo chứ. Mức lương cho vị trí này vào khoảng 1500-2500$.
Manager/Director Chúc mừng, ở vị trí này bạn đã được gọi là sếp, cấp trên, lãnh đạo,… Lúc này bạn sẽ không có thời gian mà code. Việc của bạn là họp hành, giao việc, phỏng vấn, trao đổi với các bộ phận, phòng ban, xử lý việc hành chính… Bạn còn phải đề ra định hướng phát triển của bộ phận mình quản lý, xây dựng qui trình làm việc, tuyển dụng,… Chế độ đãi ngộ cho vị trí này chênh lệch khá lớn tùy công ty và cấp bậc manager nên mình không có con số cụ thể.	**Software Architect** Muốn đạt chức danh này, ít nhất bạn phải có khá nhiều năm trong ngành. (Nhìn ai mặt mũi trẻ măng mà vỗ ngực tự xưng SA thì đừng tin). Công việc của bạn khá gian khổ: Từ một yêu cầu "mơ hồ" của khách hàng, bạn phải làm việc với BA để đánh giá solution, làm việc với PM để xây dựng một team, làm việc với Technical Lead để thiết kế, đưa ra các quyết định quan trọng về kiến trúc hệ thống. Vị trí này mặc dù không có quyền quản lý, nhưng lại có khá nhiều quyền lực ngầm.

Ngoài những con đường trên, các bạn có thể đi theo hướng Sales, Kĩ sư cầu nối (BrSE), Business Analyist. Bạn không cần phải quyết định con đường nghề nghiệp từ quá sớm. Sau khi đi làm một thời gian bạn sẽ tự nhận ra điểm mạnh, điểm yếu của mình và chọn được con đường phù hợp thôi.

Tóm tắt

- Có khá nhiều lựa chọn nghề nghiệp cho các bạn học ngành IT: developer, tester, BA.
- Mới đi làm, bạn sẽ được gọi là fresher hoặc junior. Làm việc một thời gian bạn "lên chức" developer, dĩ nhiên là cũng lên lương.
- Bạn có thể phát triển theo hướng quản lý và kĩ thuật, tùy vào khả năng và sở thích của bản thân.

LẬP TRÌNH VIÊN NÊN ĐỌC NHỮNG SÁCH GÌ?

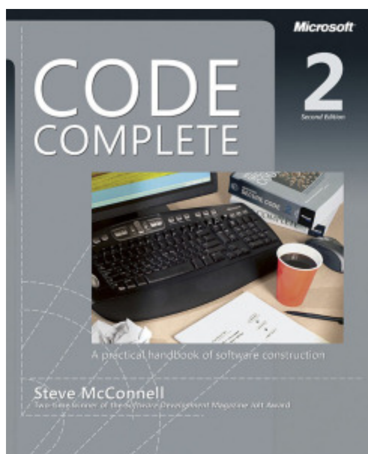
Nhiều bạn thường nghĩ rằng: developer thì cần gì phải đọc sách, code nhiều là giỏi thôi. Vâng, các cậu có cu, nhầm, các cụ đã có câu là "practice make perfect", cứ làm hoài là giỏi. Tuy nhiên, phải làm đúng cách thì mới giỏi được, code dở mà không chịu tìm cách cải thiện kĩ năng code, cứ code hoài mỗi kiểu cũ thì chẳng bao giờ giỏi được.

Mình đọc cũng kha khá nhiều sách kĩ thuật và sách về ngành lập trình, hay có dở có. Mỗi cuốn sách dù hay hay dở đều làm mình ngộ ra được vài điều. Khảo sát trong cuốn Code Complete cho thấy trung bình một developer đọc ít hơn một cuốn sách mỗi năm, ở Việt Nam chắc còn thấp hơn nữa.

Chỉ cần các bạn làm theo mình, mỗi năm đọc ít nhất một cuốn sách, các bạn sẽ giỏi hơn khoảng 90% developer còn lại rồi nhé. Bài viết này sẽ giới thiệu một số sách về kĩ thuật, về nghề nghiệp mà mỗi lập trình viên nên đọc. Các bạn có thể tìm bản ebook trên mạng, đặt mua sách từ amazon hoặc một số tiệm sách tiếng Anh nhé.

Sách về kĩ thuật lập trình

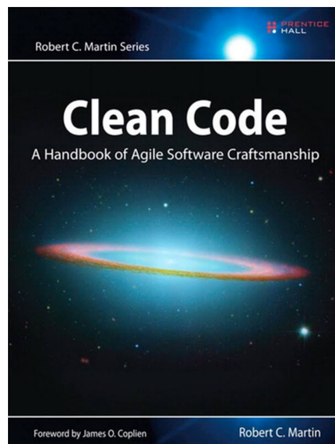
1. Code Complete



Nếu bạn muốn theo đuổi công việc lập trình một cách nghiêm túc, bạn nên đọc cuốn sách này. Mình được một ông anh giới thiệu. Có thể nói là đọc tới đâu ngộ ra tới đấy. Trong quá trình code, có lúc bạn sẽ gặp các trường hợp như: tách method thế nào, chia class ra sao, đặt tên biến thế nào, ...

Cuốn sách này sẽ là người thầy, người anh của bạn, với vô số hướng dẫn từ tổng quan như: xây dựng kiến trúc, liên hệ giữa các component, ... cho tới chi tiết như: cách tổ chức function, cách đọc tên biến.

2. Clean Code: A Handbook of Agile Software Craftsmanship



Mình đọc Clean Code trong thời gian còn làm việc ở FPT Software. Cuốn sách này xứng đáng là sách gối đầu giường của mọi developer. Mình khuyên các bạn nên mua bản gốc, vừa để đọc, vừa để phòng khi gặp đồng nghiệp code ngu, có thể cầm cuốn này đập vào đầu nó và bắt nó đọc.

Như cái tên "Clean Code", đây là cuốn sách hướng dẫn các bạn developer viết ra "code sạch". Thế nào là code sạch? Theo định nghĩa của sách, đó là code dễ đọc, dễ hiểu, dễ sửa chữa và bảo trì. Có bạn sẽ xì mũi bảo: Giời, có gì đâu, cái đấy ai code chả được. Mời bạn thử đọc lại code của một dự án mình đã làm cách đây 3-6 tháng, xem có hiểu được mình viết gì ko? Nếu không tức là code của bạn chưa được sạch lắm đâu. Cuốn sách sẽ thay đổi cách nghĩ cũng như cách viết code của bạn.

Sách về thiết kế và UI/UX

3. Don't make me think



Một cuốn sách rất hay về thiết kế giao diện. Nó đưa ra một qui tắc rất đơn giản và hữu dụng trong thiết kế UI: Người dùng rất lười, hãy thiết kế sao cho người dùng ít suy nghĩ nhất. Cuốn sách không hướng dẫn cách thiết kế đẹp mà hướng dẫn cách thiết kế đơn giản nhất và dễ sử dụng đỡ tốn công sức người dùng nhất.

Sách còn hướng dẫn một số control nên dùng khi thiết kế web: form, checkbox, radio, dropdown, … và cách sử dụng những control này hợp lý. Ngoài ra còn có một câu

chuyện về "Một button đáng giá 300 triệu đô" trong sách, về sự đắt giá của việc thiết kế giao diện[6].

4. The Design of Everyday Things

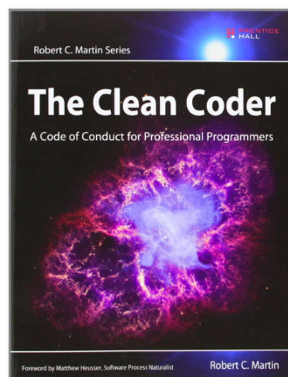


Cuốn sách này được viết bởi Don Norman. Ông là một bậc thầy về tâm lý học và design. Ông đưa ra khái niệm "user-centered design" (Design tập trung vào người dùng), chú trọng vào tính hữu dụng và dễ dùng của thiết kế, yếu tố thẩm mĩ chỉ là phụ.

Ý kiến cá nhân mình thì cuốn này khá hay, ảnh minh họa cũng rất nhiều. Sách sẽ làm thay đổi tư duy thiết kế của bạn, do đó dân design hay lập trình cũng đều nên đọc. Như tựa đề, cuốn sách phân tích thiết kế của các vật dụng thường ngày như: cánh cửa, tủ lạnh, máy lạnh, ... cho tới điện thoại, hệ thống máy tính, ... đồng thời phân tích tại sao design này thành công, design kia thất bại. Đảm bảo các bạn sẽ nhiều lần gật gù "à ra thế" khi đọc cuốn này.

Sách về nghề nghiệp và ngành lập trình

5. The Clean Coder



Cuốn sách mang tên The Clean Coder, đọc cũng giông giống Clean Code. Tuy nhiên, có một điều thú vị là nội dung hai cuốn sách lại... hoàn toàn trái ngược nhau!

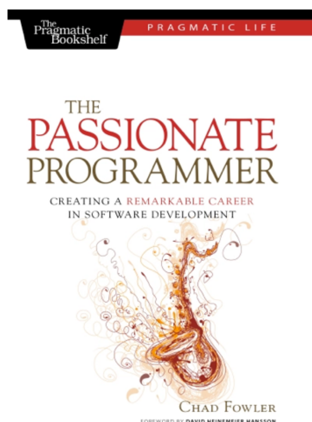
Trong khi Clean Code tập trung vào khía cạnh kĩ thuật: hướng dẫn lập trình viên cách tổ chức code và viết code sạch; The Clean Coder lại tập trung vào khía cạnh nghề nghiệp: thái độ với

[6] Đọc thêm về câu chuyện này trong bài viết "Một button trị giá 300 triệu đô"

công việc, cách làm việc nhóm, quản lý thời gian.

Trong The Clean Coder, Uncle Bob nói về nhiều khía cạnh: đạo đức nghề lập trình, ứng phó với áp lực công việc, rèn luyện làm mới kĩ năng, làm việc nhóm... Mỗi khía cạnh đều đi kèm với những trải nghiệm quí giá của chính tác giả qua hơn 42 năm tuổi nghề với đủ mọi vị trí từ dev, manager, PM cho tới CEO.

6. The Passionate Programmer



Đây là một cuốn sách viết về con đường phát triển sự nghiệp cho developer nói chung, cũng như dân IT nói riêng. Tác giả xem sự nghiệp IT như một sản phẩm hàng hóa. Để có một sản phẩm thành công, các doanh nghiệp thường thực hiện 4 bước: Nghiên cứu thị trường, đầu tư sản phẩm, phát triển sản phẩm, marketing.

Một cuốn sách khá hay với cách viết thú vị và dễ đọc. Sách vô cùng hữu dụng cho những bạn lập trình viên đang cảm thấy phân vân, cần những lời khuyên trong sự nghiệp.

7. The Healthy Programmer



Bạn có biết lập trình là một trong những ngành độc hại: Ngồi nhiều sẽ dẫn tới vô số những bệnh trầm trọng và mãn tính; lập trình viên rất dễ mắc các bệnh tim mạch, béo phì, đau khớp, ...?

Hãy đọc The Healthy Programmer – một cuốn sách viết về sức khỏe dành riêng cho developer. Cuốn này mình đọc từ hồi đầu năm khi tình cờ thấy nó trên một trang ebook. Như tựa đề, đây là một cuốn sách dành cho dân lập trình, nhưng nội dung không nói gì về lập trình mà lại nói đến một vấn đề mà lập trình viên nào cũng quan tâm: sức khỏe.

PHẦN 2 – KĨ NĂNG CƯNG CỨNG

Trong phần 1, mình tập trung nhiều vào kĩ năng mềm. Điều đó không có nghĩa là kĩ năng cứng không quan trọng. Kĩ năng mềm có tốt đến mấy mà kĩ năng cứng không ổn thì bạn cũng khó mà sống lâu trong nghề và đạt được sự nể trọng của anh em đồng nghiệp. Vì lẽ đó, phần này của quyển sách sẽ đưa bạn từ mềm đến cứng, trở nên hoàn thiện.

Vài khái niệm cơ bản trong ngành phần mềm

Phần này giải thích về một số khái niệm quan trọng trong ngành phần mềm như: Scrum/Agile, UI/UX, CI, Technical Debt, comment, design pattern,… Một số khái niệm này đã được dạy ở trường. Tuy nhiên, sách giải thích thêm bằng giọng văn hài hước, dễ hiểu, đồng thời bổ sung thêm cách áp dụng chúng vào thực tế qua các ví dụ cụ thể.

MỘT BUTTON TRỊ GIÁ 300 TRIỆU ĐÔ – CÁI NHÌN KHÁC VỀ GIAO DIỆN VÀ CHỨC NĂNG

Ngày xửa ngày xưa, có một trang web bán hàng…

Đây là một chuyện nho nhỏ, về một button nho nhỏ và một số tiền… không nhỏ chút nào. Mình đọc được chuyện này được trong cuốn Don't make me think – một cuốn sách khá hay về UI/UX[7].

Ngày xửa ngày xưa, ở một đất nước nọ, có một trang web bán hàng… Chức năng cơ bản của một trang web bán hàng thì ai cũng biết: hiển thị hàng, cho hàng vào giỏ và và thanh toán. Câu chuyện của chúng ta bắt đầu ở chức năng "Thanh toán", khi người dùng đã cho hết hàng vào giỏ, một form nho nhỏ xinh xinh sẽ hiện ra với 2 trường Tên Đăng Nhập và Mật Khẩu, 2 button Đăng Nhập và Đăng Ký. Thế nhưng, chính cái form be bé xinh xinh này đã gây thiệt hại đến 300.000.000$/năm cho trang web bán hàng.

Vấn đề ở chỗ, người dùng phải đăng nhập hoặc đăng ký trước khi muốn thanh toán. Đội lập trình nghĩ rằng "Chỉ cần đăng ký tài khoản lần đầu, thông tin người dùng có thể được lưu lại, ở những lần sau họ không cần nhập địa chỉ và thông tin thanh toán nữa. Người dùng tiết kiệm được thời gian, cũng khuyến khích được người dùng quay lại mua hàng, hai bên cùng có lợi còn gì?".

Lập trình viên đôi khi nghĩ rằng mình hiểu người dùng

Đội ngũ designer đã làm một cuộc thử nghiệm, đưa tiền cho người dùng để họ thực hiện quá trình mua hàng và thanh toán. Đối với những khách hàng mới của trang web, họ phát hiện ra một điều: người dùng rất ghét việc đăng ký, với suy nghĩ "Mình muốn mua hàng, chứ không phải muốn đăng ký đăng kiếc gì cả". Chưa kể, người dùng còn sợ bị mất thông tin cá nhân hoặc bị gửi mail spam vào hộp thư.

Với những người dùng quay lại lần thứ hai, thứ ba – đối tượng mà các developer nhắm tới, tình cảnh cũng chẳng khá khẩm hơn. Họ không nhớ được mật khẩu của mình. Mặc dù chức năng "Quên mật khẩu" vẫn hoạt động, đến tận 160.000 người dùng chức năng này mỗi ngày, 75%

[7] Đã giới thiệu trong bài "Lập trình viên thì nên đọc sách gì?"

trong số đó không tiếp tục quá trình thanh toán sau khi đã yêu cầu mật khẩu.

Chiếc form nho nhỏ xinh xinh kia, hóa ra lại là thứ ngăn cản người dùng mua hàng, rất nhiều người dùng. Thế mới biết, developer nhiều khi nghĩ mình hiểu được người dùng, nhưng thật ra không phải vậy.

Button trị giá 300 triệu đô la

Đội ngũ designer đã giải quyết vấn đề này một cách vô cùng đơn giản. Họ bỏ đi nút Đăng Ký, thay vào đó bằng nút Tiếp Tục và dòng chữ "Bạn không cần đăng ký, hãy bấm nút Tiếp Tục để thanh toán. Để thanh toán nhanh hơn ở những lần sau, bạn có thể đăng ký một tài khoản vào lúc thanh toán".

Kết quả: Lượng thanh toán của khách hàng tăng lên đến 45%. Sau tháng đầu tiên, doanh số tăng 15.000.000$. Sau năm đầu tiên, thu nhập của web tăng đến tận 300 triệu đô la. Tất cả những việc mà họ đã làm là gì? Chỉ là thêm một button mà thôi.

Ảnh minh họa

Một cái nhìn khác về UI/UX và chức năng

Xin nhắc lại một chút về UI và UX[8]:

- UI (User Interface) – Giao diện người dùng: Đây là những gì người dùng nhìn thấy và giúp họ tương tác với hệ thống (Form, button, website).
- UX (User Experience) – Trải nghiệm người dùng: Đây là những gì người dùng trải nghiệm, bao gồm cảm xúc, suy nghĩ, quá trình. UI chỉ là một phần của UX. Ví dụ như khi bạn mua hàng trên tiki. Các trang web thanh toán, web mua hàng chính là UI. Quá trình mua hàng và nhận hàng, sự thoải mái khi thanh toán, sự hỗ trợ của Tiki dành cho khách hàng chính là UX.

Trong câu chuyện trên, bằng cách thay đổi UI (Thêm một button) và không bắt đăng kí, họ đã trực tiếp cải thiện UX (Thay đổi trải nghiệm mua hàng, mua nhanh hơn mà không cần đăng ký), từ đó làm tăng doanh số bán hàng.

Hẳn nhiều bạn lập trình viên vẫn còn suy nghĩ: Chức năng mới là quan trọng, còn mấy thứ ruồi bu như giao diện thì làm thế nào cũng được (Một phần chắc là do lúc mới học lập trình toàn phải viết chương trình Console). Đây là một suy nghĩ hoàn toàn sai lầm.

Nếu chỉ viết vài tool đơn giản cho chính mình hoặc bạn bè sử dụng thì đúng là giao diện không quan trọng. Nhưng nếu đã tạo ra sản phẩm hướng tới người dùng, thì UI/UX là những thứ quan trọng nhất nhì, có khi còn hơn cả chức năng. Hãy nhớ một điều mình muốn nói qua câu chuyện này: Đôi khi chỉ một button nho nhỏ lại có giá trị đến 300 triệu đô đấy.

Tóm tắt

- Lập trình viên thường nghĩ rằng mình hiểu người dùng, nhưng đôi khi không phải vậy
- UI/UX có tầm quan trọng không kém gì chức năng của một chương trình.Hãy nhớ, đôi khi một button nho nhỏ cũng có thể có giá đến 300 triệu đô

[8] Xem lại bài viết "Tổng quát về UI và UX"

LUẬN VỀ TECHNICAL DEBT – NỢ KIẾP NÀY, DUYÊN KIẾP TRƯỚC

Technical Debt (Nợ kĩ thuật) là một món nợ mà hầu như lập trình viên nào cũng phải gánh trong quá trình làm việc. Hẳn bạn sẽ thắc mắc: Hầu hết lập trình viên chúng ta đều là những con người siêng năng chăm chỉ, không cờ bạc gái gú, hết giờ làm là đi nhậu rồi mát xa … à không, về nhà với vợ con. Chúng ta không vay mượn ai bao giờ thì làm sao có nợ???

Muốn biết câu trả lời, hãy đọc kĩ bài viết này để tìm hiểu thêm về Technical Debt nhé! Đây là một khái niệm khá quan trọng đấy.

Technical Debt là gì?

Khái niệm này được đưa ra bởi Ward Cunningham (Cha đẻ của wiki đầu tiên). Trong cuộc sống, đôi khi bạn sẽ phải mượn tiền để xài trước, sau đó cày cuốc trả. Số tiền này được gọi là nợ. Trong lập trình cùng thế, đôi khi ta chọn cách giải quyết "mì ăn liền", giải quyết được vấn đề ngay, nhưng sẽ gây khó khăn cho quá trình phát triển và bảo trì về sau. Mỗi lần như vậy, ta tạo thêm một khoản "nợ kĩ thuật" cho dự án.

Technical Debt ban đầu rất ít, nhưng theo quá trình code thì càng ngày nó càng nhiều lên, trở thành nợ nần chồng chất. Một số ví dụ:

- Để tái sử dụng code đã viết, ta copy và paste code và sửa đôi chút (thay vì phải tách thành module riêng). Cách làm này nhanh, nhưng khi có bug thì sửa… hộc máu vì code được copy ở nhiều chỗ.
- Khi có requirement mới, thay với thiết kế code cho dễ mở rộng, ta viết thêm hàm if. Cách này nhanh, nhưng nếu mở rộng nhiều thì code sẽ có một đống if.
- Có bug khủng liên quan tới kiến trúc hệ thống, thay vì fix bug và refactor thì ta try/catch nuốt lỗi và fix tạm ở phần ngọn, gọi là hotfix.

Technical Debt là điều tất yếu trong quá trình code. Mỗi quyết định ta đưa ra trong lúc code đều làm tăng số nợ này lên. Điều quan trọng là mượn xong thì phải trả, nếu để lâu, techinical debt tích luỹ sẽ gây ra nhiều hậu quả nguy hiểm khôn lường.

Tác hại "khủng khiếp" của nợ

Nếu không trả nợ, cả vốn lẫn lãi sẽ dần chồng chất trong quá trình phát triển. Quá nhiều technical debt làm chậm tốc độ của team, đồng thời ảnh hưởng đến tinh thần làm việc của các thành viên trong nhóm.

Trong nhiều dự án, vì ban đầu bị trễ deadline nên team phải code ẩu, sinh ra technical debt. Nợ làm cho tốc độ phát triển chậm dần lại, dẫn tới trễ dealine -> code ẩu -> thêm nợ, tạo thành một vòng lẩn quẩn. Một tính năng bình thường có thể chỉ mất 1 ngày để hoàn thành, nhưng nếu technical debt quá nhiều sẽ mất tới 1 tuần.

Tới một mức nào đó, khi không trả được lãi nữa, ta sẽ bị "phá sản". Lúc này, code hiện tại đã nát tới mức cực kì khó mở rộng hay bảo trì, phải đập đi viết lại. Đây cũng là nguyên nhân gây trễ deadline và dẫn đến thất bại cho nhiều dự án.



Vòng tròn lẩn quẩn: trễ deadline -> nợ -> code chậm -> trễ deadline

Nợ ơi em từ đâu tới?

Nếu như nợ công của Việt Nam là do các bác "ở trển" dùng vốn không đúng cách thì nợ kĩ thuật (technical debt) lại do chính bản thân các developer gây ra.

Có rất nhiều lý do gây ra technical debt:

- Do khách hàng thay đổi requirement liên tục, kiến trúc dự án không kịp thay đổi cho phù hợp

- Do bị deadline dí hoặc manager gây áp lực nên developer code ẩu để hoàn thành công việc
- Do bản thân developer làm biếng nên code không có comment và không viết tài liệu
- Do team không có technical lead giỏi, hoặc các thành viên không có nền tảng kĩ thuật tốt

Đôi khi technical debt là do cố ý: Chấp nhận làm nhanh để có sản phẩm giao khách hàng, giành dự án, vấn đề technical tính sau. Hoặc trong các công ty startup, người ta xây dựng sản phẩm khả thi tối thiểu (MVP) nhanh nhất có thể để khảo sát nhu cầu người dùng. Lúc này, chức năng và tốc độ phát triển mới là quan trọng nhất, code ẩu hay kiến trúc tệ cũng không quan trọng.

Làm sao trả nợ?

Như mình đã nói, code nào cũng sẽ có bug, dự án nào cũng sẽ có technical debt. Cách đối phó với technical debt là tạm ngưng việc phát triển và tập trung vào trả nợ. Ta có thể trả nợ bằng cách phân tích và tái cấu trúc hệ thống hoặc viết thêm tài liệu, viết thêm test case, refactor code để code rõ ràng hơn, dễ cải tiến hơn.

Đôi lúc ta cũng có thể bỏ qua technical debt, ví dụ như khi làm prototype để demo cho khách hàng. Vì prototype xong rồi vứt luôn nên ta có thể xù nợ. Tuy nhiên nên cẩn thận, có rất nhiều trường hợp khách hàng đòi mở rộng hoặc nâng cấp prototype thành sản phẩm để… tiết kiệm thời gian. Lúc này ta phải vắt chân lên cổ mà trả nợ luôn!



Prototype bằng giấy cho đỡ tốn công code

Hãy nhớ một điều: Mỗi lần bạn code ẩu, code đểu, bạn đang thêm nợ cho dự án. Nợ đời có vay có trả, bạn không trả thì người khác trong team sẽ trả. Technical debt phải trả bằng thời gian, công sức và mồ hôi nước mắt của lập trình viên đấy nhé.

À, nếu sắp nghỉ việc, chuyển công ty thì các bạn cứ code ẩu thoải mái, không sao đâu! Một lập trình viên "xấu số" nào khác sẽ trả nợ giùm bạn.

Tóm tắt

- Technical Debt là những món nợ về kĩ thuật, ta "mượn nợ" để phát triển nhanh hơn, nhưng sẽ ảnh hưởng đến tốc độ phát triển về sau
- Nếu không trả nợ kịp thời, nợ ngày càng chồng chất sẽ làm chậm tiến độ dự án hoặc "phá sản"
- Có nhiều nguyên nhân gây ra nợ: do khách hàng, do developer code ẩu, do qui trình không tốt
- Để trả nợ, có thể viết thêm tài liệu, unit test, refactor hoặc tái cấu trúc hệ thống

LUẬN VỀ COMMENT CODE (PHONG CÁCH KIẾM HIỆP)

Comment code luôn là vấn đề gây tranh cãi sứt đầu mẻ trán trong giới võ lâm. Xưa kia, thuở còn mài đít trên ghê nhà trường, ta thường được các thầy dặn rằng: Code nhớ phải comment. Thuở mới đi làm, sơ nhập giang hồ, mỗi khi đọc code không hiểu, ta cũng hay đập bàn mà chửi: "Thằng này code ngu quá đ*o comment gì cả".

Thế nhưng, lưu lạc giang hồ bao năm, đọc code cũng nhiều; từ code không comment cho tới code comment đầy đủ và sạch sẽ; ta vẫn băn khoăn không biết thật sự phải comment thế nào mới đúng. Thế rồi, sau khi đọc Clean Code, ta như lượm được bí tịch võ công trong truyền thuyết.

Ta ngộ ra được cái gọi là "đạo code", "đạo comment", đặt một bước chân vào con đường võ đạo đỉnh cao (Đạo ở đây là đạo lý, ko phải đạo nhạc như Sơn Tùng, các đạo hữu đừng hiểu lầm).

Lấy chuyện kiếm hiệp nói chuyện code

Thần Điêu Đại Hiệp của Kim Dung có đoạn: Dương Quá bị lạc vào kiếm mộ của Độc Cô Cầu Bại. Thuở còn sống, Độc Cô cầu bại là người kiếm thuật vô song, danh chấn thiên hạ. Trước khi chết, ông chôn 4 thanh kiếm ở nơi gọi là kiếm mộ.

Tại hạ muốn quý vị chú ý chi tiết này:

> Một hồi sau, chàng mới đặt thanh kiếm nặng đó xuống, nhấc thanh kiếm thứ ba lên, lần này chàng lại bị lầm. Chàng cứ tưởng thanh kiếm này phải nặng hơn thanh kiếm vừa rồi, nên vận lực ra cánh tay. Nào ngờ nó nhẹ tênh như không, chàng ngưng thần xem kỹ, hóa ra đó là một thanh kiếm gỗ, chôn dưới đá lâu năm, thân và cán kiếm đều đã bị mục, đọc dưới mặt đá có khắc dòng chữ:
>
> Sau bốn mươi tuổi, không mang binh khí.
>
> Thảo mộc trúc thạch đều có thể dùng làm kiếm.
>
> Cứ thế tinh tu, đạt tới cảnh giới vô kiếm thắng hữu kiếm.

Tổng kết lại, cuộc đời tu kiếm của Độc Cô Cầu Bại bao gồm ba giai đoạn:

- Lúc trai trẻ, lòng đầy nhiệt huyết mà thiếu sự chín chắn thì sử dụng tử vi kiếm là thanh bảo kiếm sắc, nhẹ và linh hoạt.
- Khi đạt độ chín của suy nghĩ và sức lực, sử dụng thanh kiếm sắt nặng nề mà không sắc bén.
- Khi bắt đầu về già, suy nghĩ và kiếm thuật đạt trình độ cao, vũ khí chỉ còn là thanh kiếm gỗ và đạt mức thượng thừa thì không kiếm mà thắng đối thủ, bất cứ thứ gì cũng là kiếm.

Ta tự thấy hầu như lập trình viên nào cũng sẽ phải trải qua ba giai đoạn này trên con đường "cầu đạo". Kẻ nào ngộ tính cao thì chỉ cần làm việc một hai năm đã đạt tới cảnh giới cuối. Kẻ nào đầu óc u mê lạc lối thì cả đời vẫn cứ ở giai đoạn hai mà thôi.

Giai đoạn trẻ trâu đầy nhiệt huyết (Code không comment hoặc comment lung tung)

Kẻ nào sơ nhập giang hồ đều trải qua giai đoạn này. Đây là khi ta còn ngồi trên ghế nhà trường hoặc vừa mới đi làm. Code chạy được là mừng, đôi khi code cho xong là nộp, chả cần comment.

Nhiều khi ta cũng nghe lời các sư phụ, thêm comment vào code. Đống comment này nhiều nhưng khá vô dụng, đọc rất buồn cười, như thể mấy kẻ mới học võ công mà bày trò múa máy bắt chước các chiêu thức cao siêu trong võ học.

```
//Chia đôi toàn bộ các phần tử của mảng đưa vào
public int[] half(int[] y){
  //Tạo một array mới chứa kết quả
  int[] x;
  //Duyệt các phần tử của mảng y
  for (int i = 0; i < y.length ; i++)
  {
     x[i] = y[i]/2; //Gán trá trị vào array chứa kết quả
  };
  return x; //Trả kết quả ra
}
```

Ta thấy đoạn code trên tuy comment khá đầy đủ nhưng chẳng có ý nghĩa mấy, thời gian viết comment còn nhiều hơn viết code. Một số coder thường dùng trò: Thêm comment để biết ngày giờ sửa code,

comment lại một số code không dùng nữa. Một số lỗi khác mà các đạo hữu này hay mắc phải như:

- Comment lại code cũ để "đề phòng". Code không dùng nữa thì xóa đi, đừng comment. Có SVN rồi thì khi cần code cũ chỉ việc restore lại là xong.
- Comment để làm log. Đừng comment theo kiểu: //02/03/1992 Sửa tên class. Sử dụng SVN có thể tra ra log rõ ràng và tiện dụng hơn nhiều.

Giai đoạn tạm chín chắn trong suy nghĩ (Biết cách comment)

Code một thời gian, hầu như mọi lập trình viên sẽ đạt đến giai đoạn này. Ở giai đoạn này, ta đã cảm thụ được một chân lý võ học : Comment nên là what, không phải how. Comment để nói code làm gì, không phải để giải thích việc code làm.

Lúc này, lập trình viên đã hiểu rõ hơn về cách comment. Với những đoạn code phức tạp, dài dòng, 1 dòng comment ngắn gọn sẽ giúp người sau dễ dàng hiểu đoạn code làm gì:

```
public object DoThing() {
  //Lấy 1 phần tử random từ database
  //Một đống code phức tạp
  return obj;
}
```

Tuy vậy, các bậc cao nhân xưa đã rút ra được một điều: Comment là thứ dối trá. Khi sửa code, comment thường không được sửa, sẽ dẫn tới tình trạng code một đằng, comment một nẻo. Ta phải đọc cả code lẫn comment để biết code làm gì, phí gấp đôi thời gian. Cách comment tốt nhất chính là dùng code, chính code sẽ nói code làm gì. Ngộ ra được điều này, các đạo hữu sẽ đạt tới cảnh giới cuối cùng trong "code học".

Cảnh giới vô kiếm thắng hữu kiếm (Vô comment thắng hữu comment)

Đây là cảnh giới mà ta hy vọng các đạo hữu có thể đạt được. Ở cảnh giới này, ta code không cần comment nhiều. Giống như Độc Cô Cầu Bại có thể lấy kiếm gỗ làm kiếm, lấy lá cỏ làm kiếm, lấy tay làm kiếm; ta thấy bất cứ thứ gì cũng có thể làm comment: tên biến cũng là comment, tên hàm cũng là comment, tên database cũng là comment. Code cũng

như comment, comment cũng như code, code và comment tuy hai mà một.

Tuy nhiên, đôi khi bắt buộc phải dùng comment: Khi viết JavaDoc, API v...v, ta bắt buộc phải comment và document, vì người dùng API chỉ được đọc document chứ không đọc code. Một số trường hợp khác: comment phải là why chứ không phải what. Ta viết comment để người khác (hoặc chính ta sau này) biết vì sao ta lại viết code như vậy. Comment có tác dụng nói những điều bản thân code không nói được. Còn việc code làm gì, chạy như thế nào, chỉ cần đọc code là hiểu.

```
// Chỉ cần nhìn tên hàm là biết hàm làm gì
public object GetRandomObjectFromDatabase() {
  return randomObj;
}

public int[] HalfAllNumbersFromInput(int[] input) {
   int[] output; //Nhìn tên biến là biết biến làm gì
   for (int i = 0; i < input.length ; i++)
   {
      output[i] = input[i]/2;
      // Viết để dubug, sau này phải remove
      // Đây là comment bắt buộc phải viết,
      // để giải thích lý do ta viết code
      Console.Write("Call this");
   };
   return output;
}
```

Ở đây, ta không phủ nhận độ hữu dụng của comment. Nhưng vấn đề là: Nhiều gã coder lợi dụng comment để viết code không rõ ràng, khó hiểu, sau đó sử dụng comment để lấp liếm. Điều ta muốn các bằng hữu hiểu là: Hãy có gắng viết code một có rõ ràng nhất có thể, bằng cách đặt tên biến, tên hàm, tách code,... Đó mới là cảnh giới tối cao của "code đạo". Đừng học đòi theo lũ trẻ trâu mà viết code kiểu "càng ngắn càng tốt", càng khó hiểu càng tốt, đó chỉ là cái dũng của kẻ thất phu mà thôi. Chúc các bằng hữu sớm thành cao thủ trên con đường cầu đạo, nhầm, cầu code của mình.

Tóm tắt

- Đừng nghĩ rằng comment càng nhiều càng tốt, mà hãy viết code sao cho dễ đọc, dễ hiểu, không cần comment.
- Comment nên là what (nói code làm gì) mà nên là why (Giải thích tại sao lại phải thiết kế, viết code như vậy).

Các khái niệm và kĩ thuật nâng cao

Phần này đề cập đến những khái niệm phức tạp hoặc ít được nói đến trong sách vở như: Nguyên lý SOLID, Dependency Injection, leaked abstraction … Bên cạnh đó là những mánh khóe, kĩ năng code, debug và những lời khuyên để nâng cao khả năng kĩ thuật. Do sử dụng nhiều thuật ngữ, khái niệm chuyên ngành nên phần này sẽ hơi khó hiểu với các bạn không thuộc ngành IT.

SOLID LÀ GÌ – ÁP DỤNG CÁC NGUYÊN LÝ SOLID ĐỂ TRỞ THÀNH LẬP TRÌNH VIÊN CODE "CỨNG"

Trong quá trình học, hầu như các bạn sinh viên đều được học một số khái niệm OOP cơ bản như sau:

- Abstraction (Tính trừu tượng)
- Encapsulation (Tính bao đóng)
- Inheritance (Tính kế thừa)
- Polymophirsm (Tính đa hình)

Những khái niệm này đã được dạy khá rõ ràng và hầu như những buổi phỏng vấn nào cũng có những câu hỏi liên quan đến chúng. Vì 4 khái niệm này khá cơ bản, bạn nào chưa vũng có thể Google để ôn lại thêm.

Giới thiệu chung

Trái ngược với 4 khái niệm nói trên, những nguyên lý được giới thiệu trong bài viết này là những nguyên lý thiết kế trong OOP. Đây là những nguyên lý được đúc kết bởi máu xương vô số developer, rút ra từ hàng ngàn dự án thành công và thất bại. Một project áp dụng những nguyên lý này sẽ có code dễ đọc, dễ test, rõ ràng hơn.

Chúng còn giúp việc bảo trì code và sửa chữa dễ hơn rất nhiều (Ai có kinh nghiệm trong ngành IT đều biết thời gian code chỉ chiếm 20-40%, còn lại là thời gian để bảo trì: thêm bớt chức năng và sửa lỗi). Nắm vững những nguyên lý này, đồng thời áp dụng chúng trong việc thiết kế và viết code sẽ giúp bạn tiến thêm một bước trên con đường thành senior (Một anh senior bên FPT Software từng dạy mình thế).

Ok, 10 phút quảng cáo đã qua, bây giờ đến phần giới thiệu. SOLID tức là "cứng", áp dụng nhiều thì bạn sẽ code "cứng". Đùa thôi, nó là tập hợp 5 nguyên tắc sau đây:

- **S**ingle responsibility principle
- **O**pen/closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle

- **D**ependency inversion principle

Trong phạm vi bài viết, mình chỉ giới thiệu tổng quát để các bạn có cái nhìn tổng quan về các nguyên lý này. Mỗi nguyên lý sẽ được giới thiệu kĩ hơn ở các bài viết sau.

1. Single Responsibility Principle

Nguyên lý đầu tiên, tương ứng với chữ **S** trong SOLID. Nội dung nguyên lý:

> Một class chỉ nên giữ một trách nhiệm duy nhất (Chỉ có thể thay đổi class vì một lý do duy nhất)

Để hiểu nguyên lý này, ta hãy lấy ví dụ với 1 class vi phạm nguyên lý. Ta có 1 class *ReportManager* như sau:

```
public class ReportManager()
{
   public void ReadDataFromDB();
   public void ProcessData();
   public void PrintReport();
}
```

Class này giữ tới 3 trách nhiệm: Đọc dữ liệu từ database, xử lý dữ liệu, in kết quả. Do đó, chỉ cần ta thay đổi database hoặc thay đổi cách xuất kết quả, … ta sẽ phải sửa đổi class này. Càng về sau class sẽ càng phình to ra.

Theo đúng nguyên lý, ta phải tách class này ra làm 3 class riêng. Tuy số lượng class nhiều hơn những việc sửa chữa sẽ đơn giản hơn, class ngắn hơn nên cũng ít bug hơn.

2. Open/Closed Principle

Nguyên lý thứ hai, tương ứng với chữ **O** trong SOLID. Nội dung nguyên lý:

> Có thể thoải mái mở rộng 1 class, nhưng không được sửa đổi bên trong class đó (open for extension but closed for modification).

Theo nguyên lý này, mỗi khi ta muốn thêm chức năng,.. cho chương trình, chúng ta nên viết class mới mở rộng class cũ (bằng cách kế thừa hoặc sở hữu class cũ) không nên sửa đổi class cũ.

3. Liskov Substitution Principle

Nguyên lý thứ ba, tương ứng với chữ **L** trong SOLID. Nội dung nguyên lý:

> Trong một chương trình, các object của class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình

Hơi khó hiểu nhỉ? Không sao, lúc mới đọc mình cũng vậy. Hãy tưởng tượng bạn có 1 class cha tên *Vịt*. Các class *VịtBầu, VịtXiêm* có thể kế thừa class này, chương trình chạy bình thường. Tuy nhiên nếu ta viết class *VịtChạyPin*, cần pin mới chạy được. Khi class này kế thừa class Vịt, vì không có pin không chạy được, sẽ gây lỗi. Đó là 1 trường hợp vi phạm nguyên lý này.

4. Interface Segregation Principle

Nguyên lý thứ tư, tương ứng với chữ **I** trong SOLID. Nội dung nguyên lý:

> Thay vì dùng một interface lớn, ta nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể

Nguyên lý này khá dễ hiểu. Hãy tưởng tượng chúng ta có 1 interface lớn với khoảng 100 methods. Việc implement sẽ khá cực khổ, ngoài ra còn có thể dư thừa vì 1 class không cần dùng hết 100 method. Khi tách interface ra thành nhiều interface nhỏ, gồm các method liên quan tới nhau, việc implement và quản lý sẽ dễ hơn.

5. Dependency Inversion Principle

Nguyên lý cuối cùng, tương ứng với chữ **D** trong SOLID. Nội dung nguyên lý:

> 1. Các module cấp cao không nên phụ thuộc vào các module cấp thấp. Cả 2 nên phụ thuộc vào abstraction.
>
> 2. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)

Nguyên lý này khá lắt léo, mình sẽ lấy ví dụ thực tế. Chúng ta đều biết 2 loại đèn: đèn tròn và đèn huỳnh quang. Chúng cùng có đuôi tròn,

do đó ta có thể thay thế đèn tròn bằng đèn huỳnh quang cho nhau một cách dễ dàng.

Ở đây, interface chính là đuôi tròn, implementation là bóng đèn tròn và bóng đèn huỳnh quang. Ta có thể chuyển đổi dễ dàng giữa 2 loại bóng đèn vì ổ điện chỉ quan tâm tới interface (đuôi tròn), không quan tâm tới implementation.

Trong code cũng vậy, khi áp dụng Dependency Inversion, ta chỉ cần quan tâm tới interface. Để kết nối tới database, ta chỉ cần gọi hàm Get, Save … của Interface *IDataAccess*. Khi thay đổi database, ta chỉ cần thay implementation của interface này.

Mình nói kĩ về nguyên lí này vì nó khá quan trọng. Nó là tiền đề để các bạn tìm hiểu Dependency Injection và Inversion of Control, hai khái niệm khó hiểu nhưng được dùng rất phổ biến trong các framework hiện đại.

DEPENDENCY INJECTION VÀ INVERSION OF CONTROL

Như đã nói ở bài trước, Dependency Injection (DI) và Inversion of Control (IoC) là hai khái niệm khó hiểu nhưng rất phổ biến trong các framework hiện đại. Mình cũng thấy vài bạn đã đi làm 1, 2 năm mà vẫn còn "ngáo ngơ" về DI, IoC, chỉ biết sử dụng nhưng không hiểu rõ bản chất của nó.

Do đó, mình viết bài này nhằm giải thích một cách đơn giản dễ hiểu về Dependency Injection. Các bạn junior nên đọc thử vì DI được áp dụng rất nhiều trong các ứng dụng doanh nghiệp và rất hay gặp khi đi làm hoặc phỏng vấn.

Phần 1: Định nghĩa

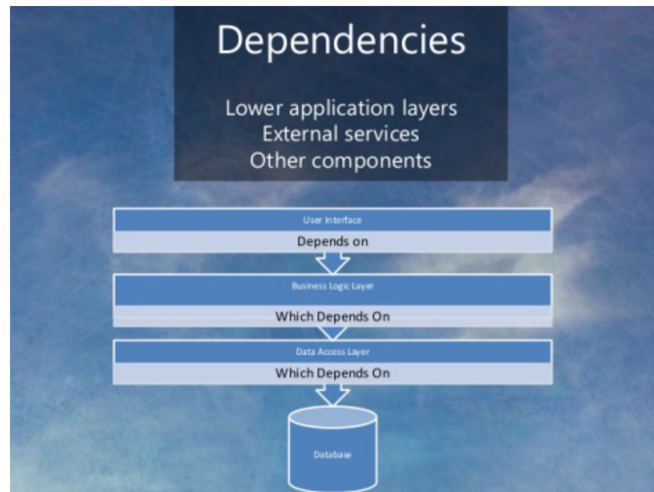
Trong bài, mình hay dùng từ module. Trong thực tế, module này có thể là 1 project, 1 file dll, hoặc một service. Để dễ hiểu, chỉ trong bài viết này, các bạn hãy xem mỗi module là một class nhé.

Nhắc lại kiến thức

Trước khi bắt đầu với Dependency Injection, các bạn hãy đọc lại bài viết về trước về những nguyên lý SOLID thiết kế và viết code. Nguyên lý cuối cùng trong SOLID chính là Dependency Inversion:

> 1. Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction.
>
> 2. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)

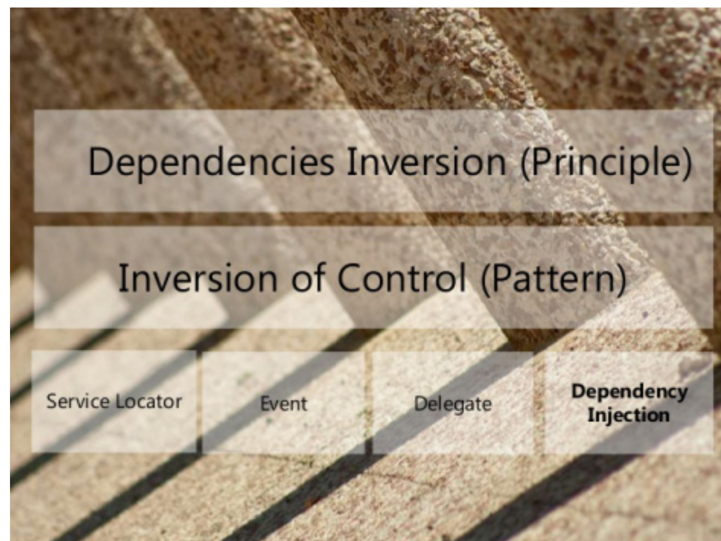
Với cách code thông thường, các module cấp cao sẽ gọi các module cấp thấp. Module cấp cao sẽ phụ thuộc và module cấp thấp, điều đó tạo ra các dependency. Khi module cấp thấp thay đổi, module cấp cao phải thay đổi theo. Một thay đổi sẽ kéo theo hàng loạt thay đổi, giảm khả năng bảo trì của code.



Nếu tuân theo nguyên lý Dependendy Inversion, các module cùng phụ thuộc vào 1 interface không đổi. Ta có thể dễ dàng thay thế, sửa đổi module cấp thấp mà không ảnh hưởng gì tới module cấp cao.

Định nghĩa và khái niệm DI

Hiện nay, các lập trình viên hay lẫn lộn giữa các khái niệm Dependency Inversion, Inversion of Control (IoC), Dependency Injection (DI). Ba khái niệm này tương tự nhau nhưng không hoàn toàn giống nhau.



Sự khác biệt giữa 3 khái niệm trên:

- **Dependency Inversion**: Đây là một nguyên lý để thiết kế và viết code.
- **Inversion of Control**: Đây là một design pattern được tạo ra để code có thể tuân thủ nguyên lý Dependency Inversion. Có nhiều cách hiện thực pattern này: ServiceLocator, Event, Delegate, … Dependency Injection là một trong các cách đó.
- **Dependency Injection**: Đây là một cách để hiện thực Inversion of Control Pattern (Có thể coi nó là một design pattern riêng cũng được). Các module phụ thuộc (dependency) sẽ được inject vào module cấp cao.

Khi nói tới DI, đa phần là nói tới Depedency Injection. Hiện nay, một số DI Container như Unity, StructureMap,… hỗ trợ chúng ta trong việc cài đặt và áp dụng Dependency Injection vào code, tuy nhiên vẫn có thể gọi chúng là IoC Container, ý nghĩa tương tự nhau.

Có thể hiểu Dependency Injection một cách đơn giản như sau:

1. Các module không giao tiếp trực tiếp với nhau, mà thông qua interface. Module cấp thấp sẽ implement interface, module cấp cao sẽ gọi module cấp thấp. Ví dụ: Để giao tiếp với database, ta có interface *IDatabase*, các module cấp thấp là *XMLDatabase, SQLDatabase*. Module cấp cao là *CustomerBusiness* sẽ sử dụng interface *IDatabase*.
2. Việc khởi tạo các module cấp thấp sẽ do DI Container thực hiện. Ví dụ: Trong module *CustomerBusiness*, ta sẽ không khởi tạo *IDatabase db = new XMLDatabase()*, việc này sẽ do DI Container thực hiện. Module *CustomerBusiness* sẽ không biết gì về module *XMLDatabase* hay *SQLDatabase*.
3. Việc Module nào gắn với interface nào sẽ được thiết lập trong code hoặc trong file XML.
4. DI được dùng để làm giảm sự phụ thuộc giữa các module, dễ dàng hơn trong việc thay đổi module, bảo trì code và testing.

Các dạng DI

Có 3 dạng Dependency Injection:

- **Constructor Injection**: Các dependency sẽ được container truyền vào (inject vào) 1 class thông qua constructor của class đó. Đây là cách thông dụng nhất.
- **Setter Injection**: Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter.
- **Interface Injection**: Class cần inject sẽ implement 1 interface. Interface này chứa 1 hàm tên *Inject*. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm *Inject* của interface đó. Đây là cách rườm rà và ít được sử dụng nhất.

```
3 references
public class InjectedCart : Cart
{
    private IDatabase _database;
    private IEmailSender _emailSender;        (new Database(), new EmailSender(), new XMLLogger());
    private ILogger _logger;

    //Constructor Injection
    1 reference
    public InjectedCart(IDatabase database, IEmailSender emailSender, ILogger logger)
    {
        _database = database;
        _emailSender = emailSender;
        _logger = logger;
    }

    //Properties Injection
    0 references
    public ILogger Logger
    {
        set
        {
            _logger = value;
        }
    }
```

Ưu điểm và khuyết điểm của DI

Dĩ nhiên, DI không phải vạn năng, nó cũng có những ưu điểm và khuyết điểm, do đó không phải project nào cũng nên áp dụng DI. Với những dự án lớn, code nhiều, cần sử dụng DI để đảm bảo code dễ bảo trì, dễ thay đổi.

Vì vậy, bản thân các framework nổi tiếng như Spring, Struts2, ASP.NET MVC, … đều hỗ trợ hoặc tích hợp sẵn DI. ASP.NET MVC từ bản 5 trở xuống cho phép ta sử dụng DI container từ thư viện, từ bản 6 thì tích hợp sẵn DI luôn, không cần phải thêm thư viện gì.

Ưu điểm	**Khuyết điểm**
Giảm sự kết dính giữa các module Code dễ bảo trì, dễ thay thế module Rất dễ test và viết Unit Test Dễ dàng thấy quan hệ giữa các module (Vì các dependency đều được inject vào constructor)	Khái niệm DI khá “khó tiêu”, các developer mới sẽ gặp khó khăn khi học Sử dụng interface nên đôi khi sẽ khó debug, do không biết chính xác module nào được gọi Các object được khởi tạo toàn bộ ngay từ đầu, có thể làm giảm performance Làm tăng độ phức tạp của code

Bài viết khá nặng về lý thuyết nên nếu bạn vẫn chưa mường tượng được sẽ áp dụng DI như thế nào vào code cũng đừng lo. Ở phần 2 mình sẽ bổ sung code minh họa, các bạn đọc xong quay lại đọc bài này sẽ dễ “thông” hơn.

BÍ KÍP ĐỂ TRỞ THÀNH "CAO THỦ" TRONG VIỆC FIX BUG

Mình từng có bài viết để than phiền về sự lười biếng, ý lại của các sinh viên ngành lập trình hiện nay [9]. Ngoại trừ một số bạn hỏi về lý thuyết hoặc vấn đề công nghệ, phần nhiều các bạn sinh viên hay lên mạng hỏi khi "gặp lỗi không biết sửa".

Qua đó, có thể thấy các bạn sinh viên năm 2 năm 3 hoặc mới ra trường vẫn còn thiếu kĩ năng debug. Bài viết này là những kinh nghiệm giúp bạn debug và đặt câu hỏi hiệu quả hơn. Mỗi khi thấy ai hỏi bài hay nhờ sửa lỗi các bạn cứ đưa bài viết này để giúp ích cho người ta nhé.

Trời đã sinh dev sao còn sinh bug?

Người ta thường bảo là developer và QA (tester) là kẻ thù không đội trời chung, một bên ráng giấu bug đi còn một bên ráng chạy chương trình để moi bug ra. Tuy nhiên, sự thật không phải vậy. Cả dev và QA đều có kẻ thù chung là bug.

Thế mới có câu chuyện rằng: Một chàng developer gặp phải con bug rất khôn, lúc ẩn lúc hiện. Chàng phải OT hết cả tuần lễ để tìm bug, không có thời gian dắt gấu đi chơi nên gấu bỏ đi theo người khác. Uất hận, chàng ngửa mặt lên trời than "Trời đã sinh dev sao còn sinh bug", sau đó hộc máu mà chết.

Thuở mới học lập trình, chúng ta thường nghĩ rằng code là chuyện khó, sửa lỗi là chuyện dễ. Bắt đầu lập trình mới biết là thời gian debug đôi khi còn nhiều hơn thời gian viết code. Thế nhưng, trường đại học lại chỉ hướng dẫn học sinh viết code chứ không bao giờ cách debug. Điều này dẫn tới việc nhiều bạn gặp lỗi nhưng không biết cách tìm lỗi cũng như không biết cách sửa.

Ở phần dưới, mình sẽ chia sẻ những bí kíp tìm lỗi từ sơ cấp đến cao cấp, cùng những điều cần lưu ý để đặt câu hỏi hiệu quả.

[9] Xem lại bài "thực trang học lập trình của các sinh viên"

Bí kiếp sơ cấp – Lỗi cú pháp

Đây là các lỗi hay gặp khi mới học lập trình, viết sai cú pháp nên chương trình không chạy được: thiếu mở đóng ngoặc, nhầm dấu bằng, thiếu chấm phẩy.

Cách giải quyết: dùng IDE xịn (Visual Studio, Eclipse, Atom). Các IDE này đều hỗ trợ nhắc lỗi cú pháp (Chỉ cần các bạn chịu khó đọc thông báo lỗi bằng tiếng Anh là được). Các lỗi này chỉ cần code nhiều là quen, hầu như code lâu sẽ hết. Bạn có thể tập một số thói quen như: Khi mở ngoặc nhớ đóng ngoặc, cuối câu lệnh phải thêm chấm phẩy.

Bí kiếp trung cấp – Exception

Sau khi sửa các lỗi cú pháp, chương trình đã build được, nhưng khi chạy lại crash hoặc quăng Exception. Exception hay gặp nhất là *NullPointerException*, khi bạn muốn truy cập vào một biến có giá trị null.

Các lỗi này cũng không quá khó xử lý. Chỉ cần đọc tên Exception và message kèm theo là bạn có thể hiểu được nguyên nhân gây lỗi. Tiếp theo, hãy copy tên Exception và message vào Google để tìm, câu trả lời thường sẽ có ở một vài link đầu tiên.

Bí kiếp cao cấp – Lỗi framework và logic

Đây thường là những lỗi phức tạp, chương trình chạy sai mà không báo lỗi hay quăng Exception gì. Ví dụ, lỗi mà bạn nào cũng gặp khi mới học là viết = thay cho ==, chương trình chạy sai mà không rõ lỗi là gì.

```
if (x=3) {
  // Do something
}
```

Các lỗi này thường khó sửa vì bạn không biết rõ nguyên nhân gây lỗi. Với những lỗi dạng này, trước tiên bạn cần xác định:

1. Code của mình sẽ chạy các bước nào, gọi những hàm nào
2. Kết quả đúng mà các hàm nên trả về là gì
3. Kết quả thực sự các hàm trả về là gì
4. Kết quả nào bị sai? Hàm nào trả về kết quả đó? Nguyên nhân sai là gì?
5. Xác định hàm chạy sai, lặp lại bước 1

Nếu chưa quen debug, hãy in ra toàn bộ các giá trị và kiểm tra xem có giá trị nào sai hay không. Khi đã quen, bạn chỉ cần đặt breakpoint, cho chương trình chạy từng dòng lệnh và kiểm tra giá trị của từng biến.

Sau khi làm rõ những điều nói trên, nhiều khả năng bạn đã tìm được câu trả lời cho mình. Nếu không tìm được hàm gây lỗi, lục tung Google nhưng không tìm ra nguyên nhân hay cách sửa, bạn sẽ phải dùng đến cách cuối cùng: Vác đi hỏi.

Tàn quyển – Làm sao đặt câu hỏi một cách hiệu quả?

Đầu tiên, hãy đặt mình vào vị trí người được hỏi, liệu khi đọc câu hỏi họ có hiểu gì không. Nhiều bạn cứ hỏi chung chung kiểu: Code không chạy được!! Thánh cũng chả hiểu code của bạn tại sao lại không chạy được. Làm rõ điều cần hỏi là bạn đã trả lời được 50% câu hỏi rồi.

Khi bạn hỏi một câu hỏi stackoverflow, bạn thường được yêu cầu chỉ post đoạn code gây lỗi lên. Hãy tập thói quen này trước khi đi hỏi: Xác định đoạn code gây bug rồi tách riêng nó ra, cố gắng tái tạo lại bug. Việc xác định được đoạn code gây bug là đã giải quyết 50% vấn đề rồi, có khi xác định xong là bạn sửa được bug luôn rồi, chẳng cần phải đi hỏi nữa.

Hãy nhớ một điều, luôn luôn Google và tìm hiểu trước khi đặt câu hỏi. Người được hỏi thường rất sẵn lòng giúp đỡ, nhưng họ sẽ rất bực mình nếu bạn hỏi những câu đơn giản mà chỉ cần 30 giây tìm Google là ra. Việc không chịu tìm hiểu hay Google trước khi hỏi chứng tỏ bạn lười và không tôn trọng thời gian của người được hỏi.

Lời kết

Kĩ năng debug cũng như kĩ năng code đều cần có thời gian rèn luyện mới có thể thành thục. Do đó, đừng buồn hay nản lòng khi bạn tốn quá nhiều thời gian để sửa lỗi. Qua một thời gian bạn sẽ quen dần và nhanh hơn thôi. Nhớ nhé, phải thường xuyên luyện tập code và tự debug thì mới nâng cao được khả năng code lẫn kĩ năng debug nhé.

Đã nói ở phần trên rồi, nhưng mình vẫn nhắc lại thêm lần nữa. Thay vì cứ gặp khó khăn là vác đi hỏi lung tung, nhớ Google 7 lần trước khi hỏi. Về lâu dài, việc này sẽ nâng cao khả năng tìm lỗi và debug của bạn đấy.

Đừng để tới lúc đi làm, cứ bí là phải lên Facebook hỏi hoặc quay sang hỏi đồng nghiệp nhé.

Tóm tắt

- Đa phần các bạn sinh viên thường thiếu kĩ năng debug – Tìm và sửa lỗi trong code của mình
- Google và Stackoverflow là bạn thân thiết nhất, hãy copy thông báo lỗi lên 2 trang này để tìm lời giải trước khi đi hỏi
- Hãy tập cách sử dụng debugger trong ngôn ngữ lập trình của mình
- Kĩ năng debug cũng như kĩ năng code, cần thời gian rèn luyện mới giỏi được

PHẦN 3 – KÍ SỰ CODE DẠO

Phần này là những câu chuyện, trải nghiệm và bài học mình rút ra được trong thời gian làm việc ở trong và ngoài nước.

TẠM BIỆT ASWIG – ĐÔI DÒNG TÂM SỰ CỦA CHÀNG JUNIOR DEVELOPER

Ngày 4 tháng 9 năm 2015, mình chính thức nghỉ việc tại Aswig Solutions để bước trên một con đường mới – du học 2 năm ở UK tại đại học Lancaster. Bài viết này là đôi dòng tâm sự kể về một năm làm việc của mình ở đây: Buồn ít, vui nhiều, học được vô số điều bổ ích về technical và cách hành xử trong công việc.

Cuối tháng 9/2014 – Bị phỏng vấn như super junior

Sau khi nộp CV vào công ty với vị trí Junior Developer, mình "được" 2 anh team leader phỏng vấn tới tận hơn một tiếng rưỡi (100% tiếng Anh). Chả biết sao vị trí junior mà mình "được" hỏi đủ thứ từ C#, Linq, MVC, Entity Framework, Database, Front-end (HTML, CSS, Javascript) cho tới OOP, Design Pattern, … riết rồi cứ tưởng tuyển siêu nhân luôn chứ không phải junior gì hết.

Sau vòng 1 căng thẳng, mình được vào nói chuyện nhẹ nhàng với anh K. Manager (Trong lòng cứ chắc mẩm là đã đậu). Đến bây giờ mình vẫn ấn tượng với câu hỏi "Em nghĩ 3 điều gì là quan trọng nhất đối với developer?". Sau này, mình mới biết vòng 2 là vòng đánh giá thái độ làm việc và tính cách nhân viên, suýt nữa phát biểu lung tung là rớt rồi.

Sau khi thỏa thuận lương, mình được hẹn phỏng vấn thêm… lần 3 với Technical Manager nước ngoài – miếng ăn đến tận miệng còn chưa ăn được, thật là khổ. Cũng may là cuối cùng mọi chuyện đều suôn sẻ, đầu tháng 10 mình bắt đầu đi làm, với mức lương khá cao so với mặt bằng junior nói chung (8 chữ số).

Công bằng mà nói, công ty khá rộng rãi với nhân viên. Trong 2 tháng thử việc mình vẫn được tính 100% lương và được khám bệnh miễn phí. Cuối tháng 12 anh Manager vẫn xét thưởng cho mình tháng 13, được gần nửa tháng lương tiền thưởng (Làm nguyên năm là được 2 tháng).

Đầu tháng 10/2014 – Chập chững vào công ty, bị ma cũ "ăn hiếp"

Ngày đầu tiên đi làm, mình được anh Phóng – một anh developer cùng team hướng dẫn cài đặt máy, chỉ chỗ ăn uống (Công ty Aswig có hệ

thống buddy, người mới gia nhập sẽ được "buddy" hướng dẫn và giới thiệu). Đến trưa, mình được anh H. Team Leader dắt đi ăn trưa miễn phí. Tính mình dễ lắm, ai bao mình ăn là mình thương liền à.

Quy định của công ty là phải sử dụng tiếng Anh trong công việc. Mới đầu mình cũng hơi ngộp vì thấy có vài bác du học về hoặc phát âm như gió, lúc họp hành thì chưa quen lắm, chẳng nghe được khách hàng nói gì. Sau 2 tuần mình mới dần tập thành thói quen và thích nghi được.

Mình lại ngồi ngay đối diện anh Manager nên không dám ho he nhiều. Ngày xưa đi làm hôm nào mình cũng ghé thăm webtretho, vozforum, kenh14… Từ hồi qua đây, sáng nào mình cũng chỉ dám vào simpleprogrammer.com, stackoverflow.com, codeproject.com, to idicodedao.com, …

Mình được vào team Foundation, một team lo việc đăng nhập và phân quyền cho các ứng dụng khác của công ty. Phase 1 của dự án không thành công lắm, và anh H. đã phải có một quyết định đau thương và táo bạo: Đập nguyên phase 1, làm phase 2 lại từ đầu. Team ban đầu có 3 người, về sau thêm được 1 người nữa là 4. Nhờ khả năng kĩ thuật xuất chúng của các thành viên cộng với khả năng chém gió "thần sầu" của anh team leader, dự án đã "tạm" thành công mĩ mãn.

Giữa tháng 4/2015 – Company trip, Thái Lan vẫy gọi

Mỗi năm công ty tổ chức company trip một lần và hầu như lần nào cũng đi nước ngoài. Kì này cả công ty được qua Thái Lan chơi, ngắm voi sẵn tiện chuyển giới luôn thể. Ở Bangkok 3 ngày nên mình cũng không đi chơi được gì nhiều. Đến hôm cuối cùng, mấy ông trong team rủ nhau đi mua quà cho vợ, ông nào ông nấy xách nguyên vali vài chục kg về. Ngẫm lại mới thấy có vợ cũng chẳng sung sướng gì.

Đầu tháng 5/2015 – Buồn thay chuyện kẻ ở người đi

Sau chuyến đi Thái Lan, nhờ khả năng ngoại giao xuất chúng, anh H. đã lấy một dự án khá béo bở về cho team mang tên Claimbook. Anh Sơn – thành viên thứ 4 của team lại quyết định nghỉ việc, qua Singapore làm… rửa chén kiêm lập trình viên cho một phòng lab ở bên đó (Nghe đồn lương cao đến hơn 4.000 đô Sing).

Anh Sơn là cựu sinh viên kiêm trợ giảng trường Bách Khoa. Khả năng kĩ thuật của anh cũng rất khá, nhưng lúc nào anh cũng bảo: Cỡ anh chỉ là cắc ké ở BK thôi. Mình nhủ thầm: Cắc ké ở BK mà cỡ này, dân BK chắc ghê gớm lắm. Tới lúc thấy ổng khoe bằng Giỏi mình mới biết mình bị lừa, hóa ra trên đời không tin ai cho được. Thiếu anh Sơn, đội ăn-nhậu mất đi một thành viên cộm cán, ai cũng buồn rười rượi….

Đầu tháng 6/2015 – Một tháng đầy biến động

Dự án thiếu người, mình bị "ép" phải dựng toàn bộ wireframe và giao diện cho dự án Claimbook. Đúng là làm super junior nó khổ thế. Một lần nữa, anh H. lại tỏa sáng với khả năng ngoại giao của mình. Anh đã "lôi kéo" được một bác Technical Lead và 2 developer mới vào team.

Anh còn dụ dỗ được team designer tham gia vào quá trình thiết kế cho dự án. Leader Team Designer là anh Sơn Đoàn – người yêu của Adrian Anh Tuấn. Trông anh Sơn đẹp trai và manly hơn ông Team Leader của mình cả chục lần.

Suốt 2 tháng trời, cả nhóm dồn hết tâm sức làm prototype cho dự án Claimbook. Team cũng tổn thất khá nhiều. Anh technical lead do thái độ làm việc không tốt, không phù hợp với văn hóa công ty, vào team mình được một tháng đã phải chuyển qua team khác rồi nghỉ việc ngay sau đó vài tuần. Design của dự án cũng bị thay đổi xoành xoạch theo yêu cầu của khách hàng, team mình cũng ngại không dám liên hệ lại với team designer vì sợ bị chửi do tự ý sửa lung tung design của họ.

Thay mặt anh H., xin gửi lời cảm ơn đến anh Sơn Đoàn đẹp trai, cùng toàn thể các bạn trong team designer đã góp phần vào thành công của dự án. Thấy team em được khen nhiều mà không nhắc đến các đóng góp của team anh nên cũng hơi cắn rứt.

Đầu tháng 8/2015 – Nghỉ việc, tăng lương và cách xử sự của người lãnh đạo

Vào khoảng tháng 5, mình cũng đã chia sẻ với anh H. Team Leader về ý định đi du học vào cuối năm để tiện cho việc sắp xếp nhân sự. Mặc dù khá tiếc nuối nhưng anh vẫn ủng hộ lựa chọn của mình.

Cuối tháng 7, đầu tháng 8, công ty bắt đầu xét tăng lương. Mấy đứa bạn mình ở team khác được tăng cũng kha khá. Anh H. Team Leader khá

khó xử khi phải giải thích với mình rằng: Do sắp nghỉ việc nên mình chỉ được thăng chức từ Junior Developer lên Software Engineer mà không được tăng lương, để phần đó lại cho các bạn khác.

Cảm thấy vừa buồn vừa hụt hẫng, mình viết một bức tâm thư gửi anh Manager và anh Team Leader (Thư dài nên mình tóm tắt nhé):

> Chào hai anh, anh H. đã có chia sẻ với em về việc em được promote lên vị trí Software Engineer và không tăng lương.
>
> Em rất hiểu và thông cảm lý do mà hai anh đưa ra quyết định này: Em cũng hiểu anh H. rất khó xử khi phải giải thích chuyện này cho em; anh K. chắc cũng phải đắn đo khi đưa ra quyết định này. Tuy nhiên, bản thân em cảm thấy rất buồn và thất vọng vì nhiều lý do sau: ...
>
> Vì vậy, em mong hai anh có thể xem lại kết quả Performance Review của em, hoặc có cách gì hỗ trợ cho em đỡ cảm thấy thiệt thòi. Cảm ơn hai anh vì đã đọc. Chúc hai anh có ngày cuối tuần vui vẻ.

Ngẫm lại, mình thấy mình kiềm chế cảm xúc khá tốt, không làm bù lu bù loa lên. Mình hiểu rằng ở vị trí Manager, có lúc phải đưa ra những quyết định thiệt thòi cho một cá nhân để giữ lợi ích công ty và tập thể. Do đó mình cố gắng lịch sự khi viết thư khi thử đặt mình vào vị trí của họ.

> Kinh nghiệm mình muốn chia sẻ là: Quan hệ và thương hiệu bản thân là những thứ rất khó xây dựng, đừng vì cảm xúc nhất thời mà hành động nông nổi rồi làm mất cả hai.

Ngay chiều hôm sau, mình được anh team lead và chị S. Manager rủ đi cafe nói chuyện và chia sẻ chân thành. Lần đầu mình được nhận lời xin lỗi chân thành của cấp trên "vì đã quên consider suy nghĩ của em khi đưa ra quyết định". Mình cũng được nghe chị S. Manager chia sẻ về việc các công ty VN đối xử không tốt với người sắp nghỉ.

Bản thân công ty cũng đã cố gắng thăng chức cho mình để khen ngợi những đóng góp của mình. Chỉ riêng việc Team Leader và Manager chịu bỏ thời gian ra giải thích, chia sẻ với junior như mình cũng làm mình cảm thấy được tôn trọng rất nhiều.

Tới hôm 1/9/2015, trong cuộc họp toàn công ty, mình được tuyên dương và tặng quà chia tay. Chưa bạn nào rời công ty lại có được đãi ngộ như vậy. Mình cũng hiểu là budget của công ty có hạn, anh H. Team Leader và anh K. Manager phải trích quỹ riêng để có quà cho mình, để mình cảm thấy được coi trọng, có thể ngẩng cao đầu tự hào khi rời khỏi công ty. Cảm ơn tình cảm và sự quan tâm mà 2 anh đã dành cho em. (Tai nghe AKG nghe sướng lắm các bạn ạ).



Kết

Hơn một năm đã qua, buồn vui cũng nhiều. Khả năng code, thái độ của mình cũng tiến bộ đáng kể. Mình cũng quan sát và học hỏi được rất nhiều về cung cách lãnh đạo, quản lý nhờ quan sát anh K., anh H. và các anh team leader khác. Cảm ơn các anh đã nâng đỡ em trên quãng đường vừa qua. ASWIG, hẹn gặp lại!

LỜI CUỐI SÁCH

Quyển sách này là tổng hợp và biên tập của hơn 60 bài viết hay nhất về nghề nghiệp cũng như kĩ thuật lập trình của Blog Tôi Đi Code Dạo[10]. Nó không chỉ là thành quả nỗ lực của chính mình, mà còn là thành quả của những bạn đọc trung thành đã theo dõi, góp ý cho blog ngay từ những ngày đầu thành lập.

Ngành IT nói chung và ngành lập trình nói riêng là một ngành khó, đòi hỏi nhiều đam mê. Kiến thức trong ngành vô cùng rộng lớn nên không thể gói gọn hết trong một cuốn sách. Dẫu vậy, mình vẫn mong có nhiều bạn sinh viên và lập trình viên mới ra trường biết tới và đọc cuốn sách này. Hi vọng cuốn sách sẽ là người bạn tốt, đồng hành cùng bạn trên những bước chân đầu tiên trong sự nghiệp lập trình.

Do thời gian và kinh nghiệm còn hạn chế nên cuốn sách không tránh khỏi sai sót, kính mong bạn đọc đóng góp ý kiến để sách có thể trở nên hoàn thiện hơn. Mọi góp ý về kĩ thuật hay cách trình bày xin vui lòng gửi về địa chỉ email: huyhoang8a5@gmail.com.

LỜI CẢM ƠN

Chân thành cảm ơn anh Đỗ Kim Cơ và ban biên tập của Tri Thức Trẻ Books đã giúp đỡ và tạo điều kiện cho mình xuất bản cuốn sách này.

Cảm ơn thầy Kiều Trọng Khánh, bác Chris Harvey, anh Lê Văn Song, anh Huỳnh Xuân Thi, anh Nguyễn Cảnh Hưng, anh Lê Anh Bảo, anh Nguyễn Thanh Bình, anh Phạm Hồng Sang đã bỏ chút thời gian trong quỹ thời gian bận rộn để đọc và viết đôi dòng nhận xét cho cuốn sách.

Cảm ơn các anh/chị đồng nghiệp tại FPT Software, ASWIG Solution và Lancaster ISS đã dìu dắt, cho mình những bài học và kinh nghiệm quí giá trong ngành lập trình.

Cảm ơn những bạn đọc trung thành của blog Tôi đi code dạo đã luôn động viên giúp đỡ mình trong quá trình viết blog. Cảm ơn hơn 1300 bạn đã hoàn thành khảo sát và cho mình góp ý về sách trước khi nó được phát hành.

[10] https://toidicodedao.com

GIẢI THÍCH CÁC THUẬT NGỮ HAY DÙNG

Ngành IT có khá nhiều thuật ngữ tiếng Anh phức tạp. Các thuật ngữ này được sử dụng nhiều trong công việc nên mình sử dụng nguyên văn trong bài viết. Điều này có thể sẽ gây khó khăn cho bạn đọc ngoài ngành hoặc các sinh viên mới.

Do vậy, phần này tổng hợp và giải thích một cách ngắn gọn các thuật ngữ hay dùng để bạn đọc dễ tra cứu. (Nếu có thời gian, mình khuyến khích các bạn tra cứu Google từng thuật ngữ để tìm hiểu kĩ hơn).

Thuật ngữ chuyên môn

- API: Viết tắt của Application Programming Interface. Có thể hiểu đây là danh sách những chức năng, dòng lệnh có sẵn mà ta có thể sử dụng.
- Bug: Lỗi của phần mềm. Quá trình tìm lỗi được gọi là debug. Sửa lỗi được gọi là fix bug.
- Build: Quá trình chuyển đổi mã nguồn (code) thành phần mềm (software)
- Code/Source Code: Mã nguồn của phần mềm. Lập trình viên viết mã nguồn để tạo ra phần mềm.
- Coder/Developer: Một cách gọi vui của lập trình viên
- Computer Science: Khoa học máy tính, một ngành học trong ngành IT thiên về nghiên cứu.
- CV: Viết tắt của Curriculum Vitae. CV được sử dụng để xin việc, chứa các thông tin như kĩ năng, kinh nghiệm làm việc
- Database: Cơ sở dữ liệu. Là nơi lưu trữ toàn bộ các dữ liệu của chương trình
- Deadline: Hạn chót để hoàn thành một dự án hoặc sản phẩm
- Debug: Quá trình tìm lỗi trong phần mềm
- Demo: Sản phẩm chưa hoàn chỉnh. Khi ta nói “demo cho khách hàng” nghĩa là giới thiệu một số chức năng cho khách hàng.
- Developer/Dev/Coder: Lập trình viên, còn gọi tắt là dev
- Fix bug: Quá trình sửa lỗi trong phần mềm

- Form: Biểu mẫu, nơi người dùng có thể nhập dữ liệu và gửi cho hệ thống
- Framework: Một bộ khung có sẵn, dựa vào đó để phát triển ứng dụng
- Git: Một hệ thống quản lý mã nguồn phần mềm
- Library/Thư viện: Tập hợp những đoạn code, chức năng đã được viết sẵn mà ta có thể sử dụng
- OT – Overtime: Chỉ việc làm thêm giờ của lập trình viên. Đôi khi có thể làm thêm tới 9-10h tối và cả thứ 7, chủ nhật.
- Outsouce: Gia công phần mềm. Sản xuất phần mềm dựa theo thiết kế, yêu cầu của khách hàng.
- Prototype: Có thể là giả lập hoặc phần mềm chưa hoàn thiện, thường dùng để mô tả phần mềm cho khách hàng
- Quora: Một trang hỏi đáp được khá nhiều lập trình viên nổi tiếng thế giới ưa thích
- Refactor code: Quá trình tinh chỉnh code, giúp code gọn nhẹ, dễ hiểu và dễ mở rộng hơn
- Requirement: Yêu cầu của khách hàng
- Server: Máy chủ. Với các ứng dụng Web, đây là nơi chứa ứng dụng và cung cấp dịch vụ
- Software: Phần mềm
- Software Engieering: Kĩ nghệ phần mềm, một ngành học trong ngành IT thiên về sản xuất phần mềm.
- Stackoverflow: Trang web hỏi đáp lớn nhất thế giới về lập trình.
- SVN: Một hệ thống quản lý mã nguồn phần mềm (tương tự git)
- Technical: Kĩ thuật. Lập trình viên thường thích dùng từ technical hơn từ "kĩ thuật".
- Test: Chạy thử chương trình. Các tester thường chạy thử chương trình để tìm ra bug.

Một số ngôn ngữ lập trình

- C#: Một ngôn ngữ lập trình được Microsoft phát triển, được dùng để lập trình các ứng dụng trên Windows hoặc ứng dụng Web, ứng dụng Window Phone.

- Java: Một ngôn ngữ lập trình miễn phí của Oracle, có thể viết ứng dụng web trên nhiều hệ điều hành. Các ứng dụng Android cũng được viết bằng ngôn ngữ này.
- JavaScript/JS: Khởi đầu là một ngôn ngữ chỉ chạy được trên trình duyệt Web. Hiện tại JS có thể dùng để viết cả ứng dụng web và ứng dụng di động.
- Objective-C và Swift: Ngôn ngữ lập trình do Apple phát triển, được dùng để lập trình ứng dụng cho hệ điều hành Mac và iOS
- PHP: Một ngôn ngữ rất phổ biến, được dùng để lập trình Web. Đa phần các trang web trên mạng được lập trình bằng ngôn ngữ này, trong đó có Facebook và Wikipedia.
- SQL: Viết tắt của SQL Query Language: Đây là ngôn ngữ được dùng để truy cập dữ liệu dưới database.

Về các vị trí trong ngành lập trình[11]

- Junior: Viết tắt của Junior Developer, chỉ các lập trình viên mới ra trường, chưa có nhiều kinh nghiệm. Đôi khi còn được gọi là Fresher.
- Senior: Viết tắt của Senior Developer, chỉ các lập trình viên lâu năm, nhiều kinh nghiệm
- Team Leader: Lập trình viên thường làm việc theo nhóm. Team Leader là người trưởng nhóm, chịu trách nhiệm phân chia công việc, hướng dẫn, đánh giá developer trong nhóm.
- Tester/QC (Quality Control): Nếu Developer là những người tạo ra phần mềm thì tester là những người kiểm tra các phần mềm để tìm ra lỗi nhằm đảm bảo phần mềm chạy đúng.
- BA - Business Analyst: Tìm hiểu hệ thống, lấy yêu cầu của khách hàng và viết thành tài liệu, giải thích cho lập trình viên

PM - Project Manager: Người quản lý dự án, chịu trách nhiệm giao tiếp với khách hàng, ước đoán tiến độ, quản lý toàn bộ các thành viên trong dự án.

[11] Đã giải thích trong bài "Con đường phát triển nghề nghiệp của developer"